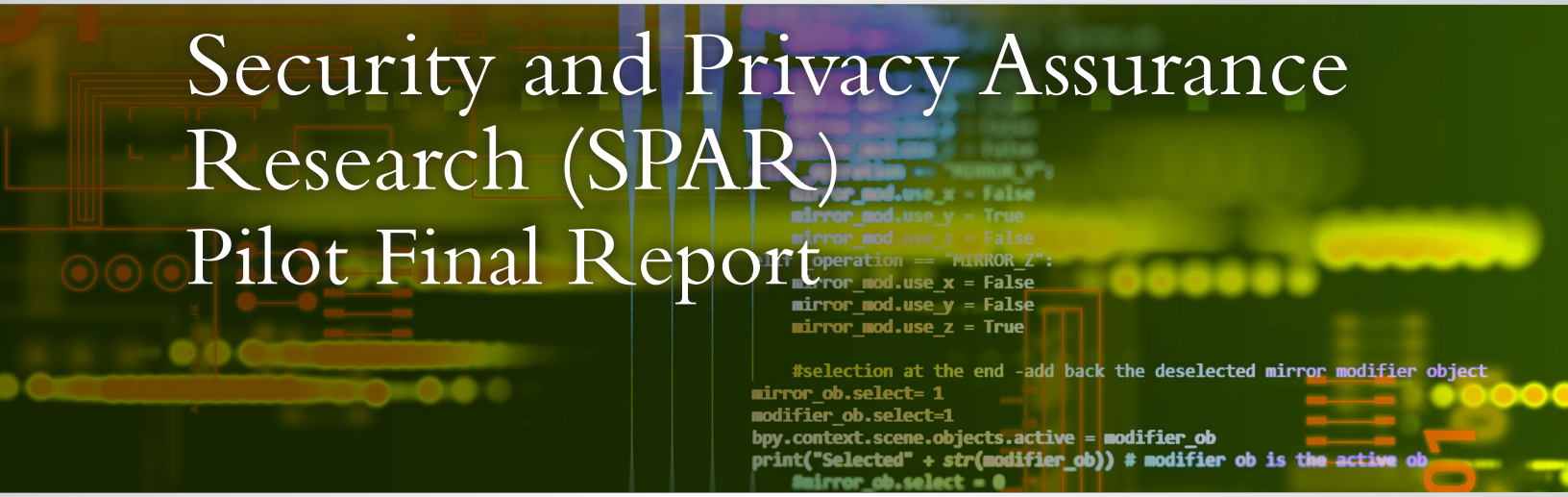
Office of the Director of National Intelligence
Intelligence Advanced Research Projects Activity

MIT LL Program 1867

Security and Privacy Assurance Research (SPAR) Pilot Final Report



DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited. This material is based upon work supported by Intelligence Advanced Research Projects Activity under Air Force Contract No. FA8721-05-C-0002 and/or FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of Intelligence Advanced Research Projects Activity.

LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

SPAR Pilot Evaluation*

Benjamin Fuller Darby Mitchell Robert Cunningham

Uri Blumenthal Patrick Cable Ariel Hamlin

Lauren Milechin Mark Rabe Nabil Schear

Richard Shay Mayank Varia Sophia Yakoubov

Arkady Yerukhimovich

entact-lincoln@ll.mit.edu

MIT Lincoln Laboratory
Lexington, MA 02420

Version 3.0
November 30, 2015

Contents

List of Figures

List of Tables

Executive Summary

Recent events have highlighted the need to balance security with dataset privacy. To improve this balance, a *data owner* and *data querier* should be able to clearly articulate what will be shared and ensure that only this information is shared. This guarantee should hold even if one of the parties misbehaves. More precisely, the parties should be able to agree on a policy for what type of questions should be asked. Then the data querier should only learn results of allowed queries and no information about irrelevant data. The data owner should be assured that the policy is properly enforced, but learn nothing about individual queries. Proper implementation of controlled and private information sharing has multiple use cases in the U.S. government. Current approaches compromise the privacy needs of either the data owner or data querier.

Cryptographic approaches can solve this problem in theory. Tools such as multi-party computation, homomorphic encryption, and symmetric searchable encryption allow parties to participate in some form of controlled information sharing. These tools have significant practicality and functionality limitations. The Intelligence Advanced Research Projects Activity created the Security and Privacy Assurance Research program or SPAR to develop practical approaches to controlled data sharing.[1]

This document describes the current state of SPAR Technical Area 1: practical and private database access. The goal is to allow the data querier to query privately a sensitive database managed by the data owner. These technologies had a very aggressive performance goal: no more than five times worse than an unprotected system. Achieving this vision would be transformative, enabling a wide variety of information sharing applications.

SPAR technology has been developed and evaluated in two settings: a research program and a pilot demonstration. The original research program developed multiple technologies with very different approaches. These technologies support various query types common in database applications. The evaluation contained two major components, evaluating: 1) the cryptographic security of the underlying technology 2) the performance of the individual query types on a software prototype. This evaluation used synthetic data and focused on performance of individual query types. Teams largely succeeded in meeting the aggressive performance and security requirements. Multiple technologies with strong security had query response times within three times of an unprotected system for at a 10 terabyte scale for important query types, faster than the goal set in the program vision.

The original research program could not predict how SPAR technology would perform in a real use case. There are crucial aspects of any use case that are difficult to predict: 1) operators' opinions on the tradeoff between privacy and performance 2) real data's effect on performance 3) how the systems would respond to realistic queries 4) what policy controls need to be enforced.

To answer these questions for a representative and important use case, SPAR technology was demonstrated in a pilot U.S. Government use case in July 2015.[2] U.S. Government opera-

[1] In Broad Agency Announcement 11-01 [22].

[2] Information about the use case can be found in [10].

tors issued a variety of queries through a web-based query specification interface. Technology limitations prevented operators from replicating some advanced tasks, but the majority of tasks were successfully completed. Operators found performance acceptable and the system usable. Based on these two evaluations, the current state of SPAR technology is:

- Security: SPAR technologies' achieve strong security with a limited amount of security imperfections. However, SPAR technologies have changed since the last security evaluation and these changes have not been formally evaluated. In addition, the impact of the security imperfections and the security of the overall system (including the software) have not been evaluated. Lastly, SPAR technology can enforce the requested policy, blocking all invalid queries; however, software errors in the enforcement mechanisms may sometimes prohibit operators from receiving responses to valid queries.

- Functionality: SPAR currently implements a large and meaningful subset of traditional query languages. During the pilot demonstration, SPAR technology was able to answer most but not all queries requested by government operators. Weaknesses of SPAR are a lack of unified query language, support for only single table databases, and inefficient support for updated records.

- Performance: SPAR implementations largely met the performance goal in the original research program. During the pilot, operators found the performance of SPAR technology acceptable. The current performance of SPAR implementations are sufficient for a low-load user facing application. Performance is inadequate for high load systems, multi-user applications.

- Software: More resources have been invested in the underlying technologies than the software implementations. The software is stable but limited. Many problems experienced during the pilot are attributable to software immaturity rather than problems with the underlying technologies.

- Usability: Multiple operators expressed that the system was easy to use and that the user interface was an improvement over current systems. However, the performance of some SPAR implementations detracted from the usability of the overall system.

SPAR technology promises to enable critical data sharing, balancing the privacy needs of the data owner and data querier. SPAR technology has been evaluated in both synthetic and real use cases. It provides strong cryptographic security, continuously growing functionality with performance adequate for deployment in user facing systems. SPAR technology could currently be deployed in a very limited use case. Additional research and development is necessary to support a rich class of use cases. SPAR technology is close to achieving the aggressive vision set in the original research program.

1 Overview

Effective data sharing is critical to the intelligence community mission. Consider the setting where a *data owner* holds a large set of sensitive data and a *data querier* wishes to see a small subset of this data. IARPA has published application parameters of anonymized use cases [1]. These use cases include internal government sharing and government/commercial sharing. In the intelligence community, there are privacy concerns for both parties. The data owner wants to protect the contents of the data set and retain control over its data. The data querier wants to hide the requested subset of data.

Simultaneously satisfying both of these privacy concerns is difficult but crucial to executing the intelligence community mission. The parties should agree on a policy for what type of queries will be answered. Then, the data querier should only learn results of allowed queries and no information about irrelevant data. The data owner should be assured that the policy is properly enforced, but learn nothing about individual queries. Data sharing technology can provide assurances that a data sharing agreement is followed. This technology should provide two types of guarantees: 1) the data is protected from outside observers and 2) the participants in the data sharing *do not* learn information beyond the data sharing agreement.

1.1 Cryptographic Approaches to Information Sharing

There are numerous cryptographic approaches to providing controlled data sharing. Tools such as multi-party computation [16], fully homomorphic encryption [11], private information retrieval [8] and symmetric searchable encryption [37] allow parties to participate in controlled information sharing. These tools have seen limited adoption due to performance and functionality limitations. The Intelligence Advanced Research Projects Activity (IARPA) oversaw development of Security and Privacy Assurance Research (SPAR) starting in 2011.[3] The goals and definition of the program are in the SPAR Broad Agency Announcement 11-01 [22]. The research program developed privacy-preserving technology in three technical areas (TA):

- TA-1: Practical Security and Privacy Assurance for Database Access
- TA-2: Homomorphic Encryption for Evaluation of Assured Data Retrieval Functions
- TA-3.1: Privacy Protection in Publish/Subscribe Systems

The common thread between these technical areas is making privacy-preserving data-search technologies fast and expressive enough for practical use. This document focuses on technology developed under SPAR TA-1: Practical Security and Privacy Assurance for Database Access (hereafter called SPAR).[4] The goal of SPAR is to allow a data querier to

[3]The SPAR program built on IARPA's Automatic Privacy Protection program (APP) which ran from 2006-2010. APP focused on single field searches for modest size data sets. APP results are included in the SPAR announcement [22, Appendix E].

[4]The two other technical areas are described in Appendix B.

ask an expressive set of queries. To satisfy the privacy requirements of both parties, they engage in an interactive protocol. The data querier should learn only the rows that match its query and no other information. The data querier should receive no data if its query was not allowed by policy. The data owner should be assured its policy was enforced.

These technologies had a aggressive performance goal: no more than five times worse than an unprotected system. This vision was designed to be difficult to meet but transformative if successful, enabling a wide variety of information sharing applications. To help met this vision, the SPAR research program introduce a separate *database server* that is independent of the data owner and data querier. While security is designed to hold against the database server, it is crucial that it does not coordinate with either the data owner or data querier. Choosing an independent party is an important part of the data sharing agreement. We discuss security provided against the database server in Section 4.1.

Members of MIT LL have written this document to describe the current state of SPAR technology. The opinions, interpretations, conclusions, and recommendations are those of MIT LL and are not necessarily endorsed by the United States Government.

1.2 SPAR Research Program

The IARPA Security and Privacy Assurance Research (SPAR) program ran from October 2011 to June 2014. SPAR's objective was to design and prototype new privacy-preserving data search technologies fast and expressive enough to use in practice. The program comprised nine research teams who worked on three Technical Areas. Phase 1 of SPAR TA-1, which ran from October 2011 to March 2013, included four research teams:

SPADE Applied Communication Sciences, Boston University, Rutgers University, University of Texas and Yale University,

Stealth Stealth Software Technologies,

BLIND SEER Columbia University and Bell Laboratories,[5]

ESPADA IBM Research and the University of California at Irvine.

The latter two teams also participated in phase 2 of SPAR from April 2013 to June 2014.

1.2.1 Program Goals

The SPAR Broad Agency Announcement (BAA) [22] tasked TA-1 teams with designing and building SQL-style database management systems that simultaneously achieve good performance at the terabyte scale along with strong security protections that minimize the information transmitted between three parties: a data owner, a data querier who wishes to query the data, and a database server that holds the owner's data in encrypted form and

[5]LGS Innovations joined and led this team during the pilot demonstration described in Section 1.4.

responds to the querier's queries. We stress here that the database server does *not* possess the decryption key(s) necessary to read the data.

These database management systems needed to allow queriers to issue SQL-style queries to a database server. The SPAR BAA requested support for up to 11 query types in total (described in Section 5). Commonly supported types include equality, range, keyword, stem, and string wildcard/substring queries, as well as boolean combinations of the above types (`and`, `or`, and threshold). Moreover, teams were required to support record-level database modifications: row insertions, updates, and deletions [44, 55].

Differentiating SPAR from existing commercial database systems is its novel set of 12 security and privacy assurance requirements. We describe these guarantees briefly here and provide a more detailed description in Section 4.2. First, other than the querier's ability to learn responses to its queries, only a small amount of information should be transmitted between entities. For example, the querier shouldn't learn about the (number or contents of) records that do not match the query, and the database server shouldn't learn the contents of database records or queriers' queries. Second, even though the data owner no longer sees the queriers' queries, she should be able to restrict the types of queries that queriers can make on her data. Third, the database contents and indexing data structures should be protected at rest using a semantically secure encryption scheme, which thwarts inference attacks that damage the privacy of property-preserving encrypted databases like CryptDB [26, 28].

Collectively, these assurance requirements request a unique balance of security imperfections vs. performance overhead. On the one hand, SPAR TA-1 systems are typically slower but more secure than commercially-available database software like CryptDB. Conversely, SPAR TA-1 systems are faster but less secure than generic cryptographic primitives like private information retrieval, secure multi-party computation, and fully homomorphic encryption (see Appendix B).

1.2.2 Major Accomplishments

In order to meet the security and privacy objectives described above, TA-1 teams designed two important new primitives. First, they designed cryptographic mechanisms that permit the database server and querier to collectively traverse an *encrypted* version of the index structure that maps search terms to records of interest. These searching mechanisms support the variety of query types described above, and they only allow a querier to learn the records that match her query. Second, they designed policy validation mechanisms that the database server can enforce *obliviously*, i.e., without ever learning the querier's query or the owner's policy rules.

Although we presented these two primitives separately, we emphasize that SPAR teams were required to *link* the two primitives to ensure that the querier submits the same query to the search and policy validation mechanisms (even though the query is encrypted, so this link is non-trivial to judge).

SPAR technologies include two main data structures 1) fast mechanisms to look up record identifiers 2) datastores with encrypted versions of individual database records. SPAR technologies link these two mechanisms so a querier can look up the relevant records based

Team	Index data structure	Main crypto primitive utilized
ACS	Prefix tree & B-tree	Conditional oblivious transfer
Stealth	B-tree	Private information retrieval
Columbia/Bell Labs/LGS	Tree of bloom filters	Garbled circuit
IBM/UCI	Inverted index	Oblivious pseudorandom function

Table 1: Table of the data structures and cryptographic primitives utilized by all four SPAR teams to perform secure index-based database searches.

on the output of the index structure. Crucially, the querier is only able to decrypt records that were output by the index structure. Throughout this report we focus on the indexing mechanisms as they are the main technical advance in the SPAR program.

While all teams were given identical requirements, they designed strikingly different cryptographic protocols. Table 1 demonstrates the variety of data structures and cryptographic building blocks that the four teams utilized in their encrypted indices (Section 2 summarizes the approaches). This variety strongly benefits potential end users of SPAR technology because the designs provide several (sometimes subtle) tradeoffs between security and performance during the execution of queries and updates. These tradeoffs exist along multiple dimensions, for example, the degree to which the secure protocols followed cryptographic standards and the degree to which performance can be boosted in a distributed cluster through parallelism.

Furthermore, teams demonstrated the viability of their designs by developing software that responded to queries. The SPAR BAA required implementations to respond to queries within a 10× performance factor of a non-privacy-preserving system on a 10 terabyte database with slow update rates. Teams largely exceeded this goal, answering many queries within a three times factor of an unprotected baseline during testing. In summary, SPAR technologies met a strong set of assurance requirements at performance levels that do not exist in other systems. MIT LL believes that the introduction of an independent database server is a crucial enabler of SPAR technologies' performance.

1.2.3 Evaluation Results

At the end of each phase of the research program, MIT LL evaluated each SPAR technology. Assessment reports have been delivered to IARPA [46, 49, 51, 54, 56, 57]. This document and previous reports are based on two evaluations of SPAR technology: 1) a review of their cryptographic algorithm's security guarantees to validate that they met the BAA's assurance requirements and 2) an empirical assessment of their software implementation's correctness, functionality, and performance to determine if they met the BAA's efficiency requirements.

Security For the security review, MIT LL first formalized the BAA's 12 assurance requirements (and the intent behind them, as determined through discussions with IARPA) into concrete security definitions laid out in our Rules of Engagement [44,55]. These requirements

are described informally in Section 4. These formal definitions were flexible enough to permit the teams to research different technologies (see Table 1) rather than enforcing a monolithic solution. At the end of each phase, MIT LL extensively reviewed the teams' proofs and arguments using a peer review-style system in order to (1) verify their correctness, (2) validate that they met the security requirements laid out in the BAA and our Rules of Engagement, (3) analyze their applicability toward US government use cases of interest, and (4) explain their operational impact in such use cases in order to help potential end customers decide which technology is best suited to their needs. Additionally, we supplemented the human-validated security assessment with a formal methods review of one team's algorithm using a tool called EasyCrypt [2] that uses a satisfiability solver to verify cryptographic arguments mechanically.

Correctness, Functionality, and Performance For the empirical assessment, each team spent one week at MIT LL at the end of each phase, during which time MIT LL precisely and comprehensively measured the performance and functionality of their software prototype on an isolated cluster. Examples of metrics collected include: correctness, latency, and throughput of query responses as a function of the query type and number of results returned; correctness and performance of the policy validation mechanism, as a function of the complexity of the policy; and correctness and atomicity of database modifications [44,45,62]. Our tests were conducted on a set of synthetically-generated databases, queries, and updates that MIT LL crafted together in order to produce clean result data that can be extrapolated to situations that our test didn't originally consider.

Collectively, teams performed very well during testing: each team met most of the assurance requirements laid out in the BAA (falling short on some requirements but exceeding expectations in others), and teams at the end of phase 2 were able to answer many queries within a three times factor of a non-privacy-preserving database at a 10 TB scale. We describe more details about functionality and performance results in Sections 5 and 6, respectively. Additionally, we refer interested readers to the full reports to the government at the end of each phase for even more information [46, 49, 51, 54, 56, 57].

1.3 Limitations of Research Program

The SPAR research program produced four technologies with different strengths and weaknesses. At the end of the research program, it was possible to compare and constrast the security of the underlying cryptographic technologies. The evaluation used synthetic data generated along with queries that produced a varied number of results. This evaluation enabled prediction of system performance for query types based on the size of the data set, selectivity of individual terms of the query, and the number of results.

Even with these experimental results it is difficult to predict in the applicability of SPAR technology in actual use cases for several reasons. These reasons are listed in Figure 1. Figure 1 is not meant to criticize the SPAR research program. Limitations 1-5 are difficult to overcome without specific use cases. The answers to these questions may vary drastically

1. The mix of users was not known. It was unknown whether users would find the functionality and performance sufficient to accomplish their tasks.
2. Testing focused on predicting performance of individual query types. It was unknown what types of queries operators issue. It was also unknown how the systems would respond to queries that combine query types. Furthermore, the robustness of each implementations' query parsing was not known.
3. The synthetic data generated was clean. It contained no NULL values and the distributions of individual fields were understood and communicated to teams.
4. Teams knew the schemas that would be tested months ahead of time. It was unknown how much the teams optimized towards these schemas.
5. Policies tested during the research program were synthetic. It was unknown what types of query policies would be necessary in real use cases.
6. SPAR implementations' were configured by their respective development teams. It was unknown how much manual tuning affected system performance and if the systems were mature enough to be deployed by individuals outside the teams.

Figure 1: Limitations of the SPAR Research Program. These limitations made it difficult to predict the utility of SPAR technology in use cases.

between use cases. Limitation 6 could have been addressed in the research program, but was not a priority. The pilot study presented in this document was intended to address these limitations.

1.4 SPAR Pilot Demonstration

Following the SPAR Research program, IARPA identified SPAR as a promising technology for transition. As listed in Figure 1 there were many unknowns about how SPAR technology would perform in actual use cases. IARPA decided to demonstrate the technology in a U.S. government use case in July 2015, which we call the *pilot*. This demonstration included the BLIND SEER, ESPADA, and Stealth technologies from the SPAR research program. This choice was made by IARPA who also selected an important use case and coordinated with the data owner and data querier. We defer discussion of this use case to the addendum of this report [10]. This pilot was designed to address the limitations in Figure 1. The objectives listed in Figure 2 are each designed to address the corresponding limitation in Figure 1.

As before, MIT LL evaluated each technology in the pilot environment and wrote software necessary to perform this evaluation. The design of this evaluation is described in Section 3.2. This document provides an updated assessment of SPAR technology using evidence from

1. Understand the users' opinions about SPAR technology. Is current functionality and performance sufficient to accomplish their workflow?
2. How would the performance of SPAR technology change with a represent set of queries combining different SPAR query types? How robust are SPAR implementations to unknown query syntax?
3. How would the performance of SPAR technology change with real use case data? Real data is often incomplete and has unique distributional properties.
4. How efficiently do SPAR implementations handle new schemas and data distributions?
5. Are SPAR technologies capable of enforcing the desired policy?
6. Is it possible for outside personnel to configure SPAR implementations?

Figure 2: Main objectives of the SPAR Pilot Demonstration. These objectives are designed to provide more information on the limitations of the SPAR Research program identified in Figure 1.

both the research program and pilot demonstrations. This assessment is organized around by the major aspects of the technology: security (see Section 4), functionality including policy enforcement (Section 5), performance (Section 6), software maturity (Section 7), and usability (Section 8).

SPAR teams developed new features during the pilot demonstration. These impact the overall security and performance of each technology. MIT LL did not perform a security assessment on the features added during the pilot. Furthermore, empirical testing performed during the research project has not been replicated on added features. The pilot evaluation was focused on a single use case and was not as extensive as the research program evaluations.

1.5 Pilot Results

U.S. government personnel found SPAR technology usable and the performance acceptable (see Section 8).[6] SPAR technology performed slower than their current systems. However, they felt that SPAR technology could provide them with access to new data sources that are not currently available. MIT LL concurs with this assessment. It would be difficult to install SPAR in a current data sharing application as users would only experience inferior performance and the privacy gains would be invisible. However, users are willing to accept

[6]Users interacted with SPAR technology through a web-based user interface designed by MIT LL. During the pilot environment, sensitivity concerns prevented operators from interacting with a control unprotected database application. It is possible that users' positive opinion is largely due to the user interface. This issue is discussed further in Section 3.2.

degraded performance if it grants new, novel intelligence sources.

During and after the pilot demonstration, MIT LL gathered information about the security, functionality, performance, usability and software maturity of SPAR technologies and implementations:

- Security (Section 4): The security evaluation in the pilot focused on the current capacity of SPAR implementations to enforce data controls. The pilot did not cover cryptographic security of the underlying technology nor the impact of their security imperfections. Each implementation was able to enforce some but not all of the desired data-control policy. MIT LL believes that these limitations are implementation issues and not due to defects in the underlying technology.

- Functionality (Section 5): Operators were able to issue a large variety of queries using query types commonly available in unprotected systems. SPAR technology was capable of executing the majority of queries requested by operators. However, multiple operators asked to execute queries that are not currently supported by SPAR technology. The pilot was conducted using static data, but inserts of up to $1,000,000$ records were performed after the pilot. Additional work is needed to support millions of new records per day in the demonstrated use case.

- Performance (Section 6): Operators found the performance of SPAR technology acceptable. The technology performed on average slower than an unprotected system but occasionally outperformed an unprotected system. Interestingly, operators found consistent response times to be more important than the mean response time. Being able to predict how long a query would take helped operators to plan their workflow and incorporate the technology into their workflow.

- Software (Section 7): The pilot demonstration was the first time that SPAR technology was deployed by individuals outside of the respective development teams. SPAR software implementations are stable but less mature than the underlying cryptographic technology. Numerous problems experienced during the pilot are attributable to software defects. The pilot gave the teams valuable feedback that will mature their software.

- Usability (Section 8): Multiple operators expressed that the system was easy to use and that the user interface was an improvement over current systems. However, the performance of some SPAR implementations detracted from the usability of the overall system.

This demonstration included three SPAR technologies. The body of the document compares and contrasts the three approaches, more detailed results for each team are included in Appendix F. MIT LL believes that each evaluated technology has different strengths and weaknesses and none is clearly stronger than the others. The optimal choice depends on the specific use case and the priorities of stakeholders. Throughout this document, to avoid

giving the impression of an explicit order between technologies, they are always presented in alphabetical order. Also we refer to each team's underlying technology and their current software implementation by the same name. However, we distinguish between problems in the underlying technology and the software implementation.

BLIND SEER BLIND SEER was the only SPAR implementation that successfully rejected all queries that should have been rejected based on policy. However, it also rejected a large number of valid queries, likely due to software parsing limitations. How quickly BLIND SEER responded to queries depended heavily on the entropy of data fields and network characteristics. BLIND SEER had a very large variability in query-response time, with some queries taking hours to complete. This variability made it difficult for operators to use the system because they did not know when queries would complete. Additionally, there were some limitations in parsing queries, but these problems appear to be fixable and not related to the underlying technology. The BLIND SEER team describes their experience in [21].

ESPADA ESPADA quickly responded to queries in a specific form called ESPADA normal form. However, responses were much slower for queries not in this form. This led to ESPADA having significant variation in query-response times, based on the form of the queries. ESPADA had the fastest "best-case" performance of the three SPAR implementations and optimized well for specific types of queries. ESPADA's policy enforcement focused on the structure of the query. This meant that ESPADA was not capable of enforcing all policy rules. The ESPADA system had strong atomicity properties, resulting in a slower update rate than other systems. Lastly, ESPADA had some parsing issues, though these appeared to be fixable. The ESPADA team describes their experience in [34].

Stealth The Stealth software was easy to setup and configure, and was stable and accurate throughout the demonstration. How quickly Stealth handled queries was largely determined by the number of records being returned. Its performance was consistent and fast throughout the pilot. Stealth's ability to reject queries based on policy was based on black-listing, and therefore Stealth was not capable of enforcing all of the requested policy rules. The Stealth team describes their experience in [38].

1.6 Current Status of SPAR Technology

The SPAR research program and the 2015 SPAR pilot demonstration are the important steps towards a mature, fieldable capability for SPAR. The SPAR pilot demonstration informed the reception SPAR would likely receive from a user community as well as the challenges of deploying SPAR in an operational context. The security of SPAR technologies was evaluated and verified in the SPAR research program.

SPAR technology has taken steps away from being "researchware" and towards being a mature technology ready for an enterprise deployment. During the pilot demonstration,

SPAR implementations were deployed by MIT LL instead of its operators, tested on an unknown and real data schema, and supported real operators queries successfully. It is possible that current SPAR implementations could be sufficient for a limited use case. Such a use case would be subject to a number of constraints, including:

- A single table schema, since SPAR does not currently support joins. (Teams have begun working on this problem [21, 34].)
- An immutable schema. Unlike traditional relational databases, SPAR does not support schema changes after ingest.
- How each field might be included in queries must be anticipated and immutable indices must be specified prior to ingest.
- A simple policy to enforce that is applied to all users.
- Tolerance for functionality and performance degradation when policy enforcement is enabled.
- Modest insert rate (all implementations struggle to support update rates of millions of records per day).
- High tolerance for performance degradation as additional records are inserted, since there is no mechanism to integrate inserted records into the primary data structure used by each implementation.
- Authentication is either not required, or could be implemented in a user interface "front end." (It would be preferable if it were integrated with cryptographic protocols.)
- Access control either enforced at the network level (e.g., IPSEC) or implemented in a user interface "front end" (It would be preferable if it were integrated with cryptographic protocols.)
- Only one user at a time, since the software doesn't currently support concurrent querier connections.
- Specialized IT support requirements to deploy and maintain the system that are not consistent with customary and/or best IT practices for enterprise systems.
- Depending on the technology selected, interactive real-time queries may be very difficult to integrate into operator workflow due to the variability of query response times.
- Limited support for auditing of queries and results.

This list describes current limitations of SPAR technologies (and their corresponding implementations). While this list of limitations is long, it represents a fraction of the challenges that existed when the SPAR program began. Some of these problems have well understood

engineering solutions. Some problems require significant new research. If there was an urgent national need for an immediate deployment of SPAR technology, it would be possible to field this capability subject to the above constraints. However, to make SPAR technology more generally applicable, additional work is necessary to resolve the remaining research and engineering challenges. In the next section, we identify improvements which we believe will have the most impact on the readiness of SPAR technology focusing on the improvements needed to support the use case of the 2015 SPAR pilot demonstration.

1.7 The Path Forward

1.7.1 Recommended Extensions

The SPAR pilot identified two major shortcomings in the current state of SPAR technology. The first is the support for multiple simultaneous policies. These policies should be enforced based on user identity. The second is the ability to support high modification rates in an efficient manner. These two extensions are necessary for any sustained deployment. While these extensions were identified as a result of the 2015 SPAR pilot demonstration they are much more broadly applicable, and would be essential to any enterprise deployment of the technology. To fulfill these extensions we recommend the following extensions:

For Sustained Deployment

Query language Current SPAR systems systems do not answer queries according to a well-defined standard. Different implementations have different parsing limitations. During both the research program and pilot demonstration, these limitations led to SPAR implementations interpreting queries in ways that were not anticipated by MIT LL (see Section 5.2). For SPAR to be widely used, it is necessary to have a well-defined language for what type of questions can be answered. Ideally, this language should be based on a well defined standard such as SQL.

Query policy language Currently SPAR systems have different policy capabilities and specification. Furthermore, specifying SPAR policies is a manual and arduous process. As an example, specifying three simple business rules required over 600 rules in the ESPADA implementation. A uniform policy specification language and process will greatly simplify SPAR deployment.

Multi-user support Current SPAR technologies are designed for a single querier interacting with a single encrypted database. In order to deploy the technology, the underlying cryptography must support multiple queriers accessing the machine simultaneously. This also means users must have cryptographically restricted views of the overall structure. This involves significant changes to underlying SPAR technologies and implementations. Different approaches may be more or less amenable to these changes.

Multi-policy support Current SPAR implementations support a single policy. With the addition of multiple queriers accessing a SPAR database it is important to correctly identify each querier and set the policy based on the querier's identity.

Auditing SPAR query policy is used to ensure proper query behavior. However, auditing of querier and owner behavior is necessary to satisfy policy. Developing specialized auditing for SPAR is important as most behavior is private. The primary purpose of auditing is to provide oversight with a way to verify the cryptographic protections of SPAR and to provide confidence their policy specification was complete.

Improved Update Support Current SPAR implementations place updated records into separate less efficient data structures. The pilot use cases for SPAR technology has a high update rate where a large fraction of data is inserted (or replaced) each day. SPAR technology needs to improve support for updates, inserts and deletes.

Simplification of software deployment Current implementations are deployed using custom scripts that require multiple manual steps. Software should be deployed through package management (apt-get, yum) that can download any dependencies, install the product and then start it with a default configuration.

Necessary to Support Advanced Use Cases

Name Variations/Fuzzy Name Matching Real database applications are error prone and data is often erroneous. Names often have multiple spellings and are entered incorrectly. To support real data, SPAR should develop query types including 1) stemming optimized for names 2) fuzzy matching specialized towards names.

Support for Relational Data Current SPAR technologies assume a single table database. Many use cases have relational data and queries combine these data sources in interesting ways. Research is necessary to provide secure implementations of database joins.

1.7.2 Remaining Evaluation

The pilot demonstration decreased uncertainty about deploying SPAR technology in the demonstrated use case. However, there are several aspects of SPAR technology that are still unknown. MIT LL recommends the following evaluation be performed before SPAR technology is deployed for a sustained period of time.

Verification of new features The security of SPAR technologies was evaluated at the end of the SPAR research program. All teams have developed significant new features whose security is unknown. The security of these features must be evaluated. In addition, the empirical evaluation of SPAR implementations should be updated to include all current features.

Configuration SPAR technology has only been configured by subject matter experts (see Section 7). The usability of SPAR implementations' configuration should be evaluated.

This involves working with system administrators to setup and configure SPAR for a real use case. Consideration should be given to the implications of incorrect configurations. Additionally, administrators should configure implementations using multiple hardware platforms to determine resource bottlenecks.

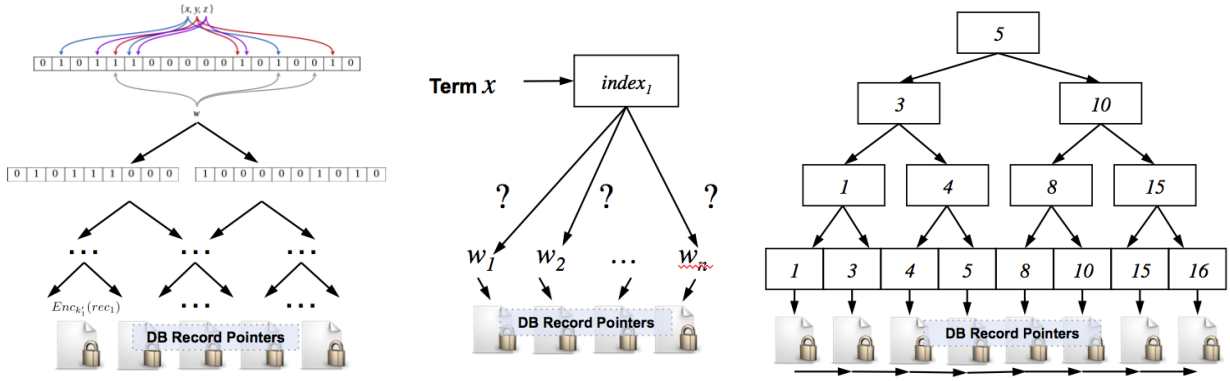
Impact of leakage SPAR technologies have several security imperfections (see Section 4). These security imperfections allow malicious actors to learn some statistics about the underlying data. A malicious actor may be able to combine information gained from these statistical priors with outside information to compromise privacy guarantees. The impact of these imperfections is application dependent but should be studied prior to a sustained deployment.

System security MIT LL evaluated the design of SPAR underlying cryptography. A full SPAR system should be evaluated too including the software itself. Software should be checked for consistency with described protocols and general software security.

1.8 Organization

The remainder of this document is organized as follows. Section 2 provides a brief overview of the BLIND SEER, ESPADA, and Stealth approaches. Section 3 describes the test design and desired metrics for both the research program and the pilot demonstration. Sections 4, 5, 6, 7, and 8 describe the current state of the security, functionality, performance, software maturity, and usability of SPAR technology respectively.

In the appendices, we provide supplemental information. Appendix A provides a glossary of terms used in this document. Appendix B describes the other SPAR technical areas that were not evaluated in the pilot. Appendix C lists the datasets and queries used in the SPAR research program. Appendix D describes the unclassified environment used to prepare the implementations for the pilot demonstration and some testing after the pilot demonstration. Appendix E compares the testing that was performed in the SPAR research program and the SPAR pilot. Appendix F provides additional detail on the pilot demonstration results for BLIND SEER, ESPADA, and Stealth. Lastly, Appendix **??** contains documents that are referenced in this report.



(a) BLIND SEER (b) ESPADA (c) Stealth

Figure 3: Visualization of SPAR technology indexing approaches. a) BLIND SEER builds an encrypted tree of Bloom filters that can be jointly traversed by querier and database server. b) ESPADA looks up set of records matching first term using inverted index. Each record that matches first term is then checked against remaining of Boolean formula using forward indices. and c) Stealth creates a separate B-tree structure for each searchable field. Tree is jointly traversed by querier and database server to find pointers to encrypted records.

2 Synopsis of SPAR Technologies

This section provides a brief introduction to the underlying approach of the three SPAR technologies discussed in this report. We concentrate on the primary index structure of each technology. The high level design choices described in this section effect all aspects of the technology. This section has two goals: 1) review of the basic approach and capabilities of each technology and 2) provide enough context to interpret the results in the following sections. Figure 3 contains a visualization of each approach to be used as a reference for the rest of this section.

2.1 BLIND SEER

We now present a brief overview of the BLIND SEER approach. More detailed information can be found in academic publications [9, 27], BLIND SEER technical reports [21, 39], or the MIT LL evaluations produced in the SPAR research program [49, 56]. BLIND SEER constructs an index consisting of an encrypted search tree. The leaves of this tree correspond to the individual records and contain all the searchable keywords from that record. Internal nodes contain a Bloom filter storing the set of all keywords stored in their descendants. A boolean query (e.g., A AND B) can be executed as follows:

- Beginning at the root, check to see if the Bloom filter of the current node contains the keywords being searched for in the query. If not, terminate the search.
- If so, visit all the children of the node recursively for the same check until the leaf nodes are reached.

- Return all leaf nodes containing the searched for keywords.

To additionally support range queries, the set of keywords in the leaf nodes is extended to include special range keywords. Negation is supported by creating ranges of values excluding the negated item.

Performing the queries as described above would leak both the query and the data to the server performing the search. Instead, the Bloom filters are encrypted to protect their content and the Bloom filter check is performed using Yao's Garbled Circuits [65] between the client and server. This allows one to check membership in a Bloom filter while hiding the content of the Bloom filter and the value being sought. Now the client and server can together traverse the search tree as before, with the server only learning the search pattern of which tree nodes are visited by the search, but not the keywords contained in those nodes or the search terms.

BLIND SEER also supports modifications (which includes inserts, deletes, and updates) within their secure index. Inserts are handled creating a separate blank Bloom Filter tree. This approach limits the total number of insert operations that can be performed. Deletions are handled by marking the corresponding nodes as deleted. Updates are implemented as an insertion and a deletion.

2.2 ESPADA

This section provides a brief overview of the ESPADA approach. More detailed information can be found in academic publications [6, 7, 23], ESPADA technical reports [30–33], or the MIT LL evaluations produced in the SPAR research program [51, 57]. ESPADA's primary structure is an expanded inverted index. In the basic scheme, without factoring in security, they store two different indexes. The first is an inverted index which maps between keywords and document identifiers and the second is a index between document identifiers and keywords contained within. In order to do a conjunctive search, the server looks up in the first index for the keyword in the conjunction that has the least matching documents. For each of the documents that matches the keyword, the server checks to see if the rest of the terms of the query are matched. If they do, that particular document is returned as part of the query. To prevent leakage about the data stored to the server, the records are encrypted and the document ids permuted using a pseudo-random function. The keys are only given to the server to decrypt the entries in the second index if they are being searched for.

2.3 Stealth

This section presents a synopsis of the Stealth approach. More detailed information can be found in Stealth technical reports [40–43], or the MIT LL evaluation produced in the SPAR research program [54]. Stealth's primary data structures are B-trees and linked list. A B-tree and linked list is constructed for every data field. We focus on searching over a single data field. We describe how to perform a range search (all other queries are expressed as range searches). The leaves of a B-tree point to records in the corresponding linked list. The

linked list contains pointers to the encrypted records of the table. The B-tree is traversed with the start of the range to obtain a pointer in the linked list. This pointer is the start of valid records for this query. The B-tree is then traversed again to obtain a second pointer in the linked list. This pointer is the end of valid records for this query. To hide the client's queries and ensure the client only obtains proper records, the B-trees are traversed using private information retrieval and the linked list is traversed using a shared-input, shared output pseudorandom function.

Stealth technology only handled searching over a single field during the SPAR research program. Support for Boolean queries was added during the pilot program. Stealth provided MIT LL with a short summary of this protocol. Each B-tree is searched separately and the records are combined using private set intersection and set union protocols.

Updates are handled by constructing separate tree structures for updated records. New records are handled by batch processing, simplifying the construction of the data structures.

3 Test Format and Methodology

This section describes the methodology for the assessment in this report. As described in the overview, this report is based on two separate evaluations: (1) the original research program and (2) the pilot demonstration. This section describes the methology for both tests.

3.1 Research Program

During the TA-1 research program, SPAR teams designed novel cryptographic algorithms to perform privacy-preserving database searches of common SQL queries at scale. At the end of each phase of the program,[7] MIT LL conducted two major evaluations of their technology.

First, MIT LL formalized the informational security goals in the SPAR BAA. SPAR teams proved security of their protocols with respect to these formal security goals. Then, SPAR teams produced complete written descriptions of their cryptographic algorithms and proofs. Lincoln analyzed these documents in detail following a peer-review style with multiple readers per report. Our goals were to ensure that the proofs were correct (akin to a journal review) and also that the proved guarantees meet the lower bounds specified in the BAA [22] and Lincoln's Rules of Engagement [44, 55]. Our final reports to the government at the end of each phase provide guidelines as to the best fit for each technology within envisioned US government use-cases. We defer more details about our security evaluation to Section 4.

Second, SPAR teams provided demonstration software to provide evidence that their technology was fast and expressive enough for tech transition. MIT LL evaluated these prototype implementations for functionality (Section 5), correctness (Section 5.2) and performance (Section 6). In this section, we catalog the metrics captured during the testing periods, the test methodology we followed to acquire these metrics, and the synthetic data used in our tests.

3.1.1 Metrics Gathered

In this section, we describe the specific metrics gathered during query and policy testing.

Query tests This section describes the procedure by which we evaluated the correctness and performance of queries. The query set used for testing is described in Appendix C. To isolate the running time of a query from that of a policy authorization, all tests in this section operate without policy enforcement (i.e., using a policy that simply accepts all queries).

First, we define two metrics for the correctness of queries. *Query precision* is the probability that a record matches the query given that it was returned. It is calculated as the number of true positives (records that match the query and are correctly returned by the SPAR implementation) divided by the total number of records returned by the implementation. Additionally, *query recall* is the probability that a record is returned given that

[7] Specifically, phase 1 testing occurred during December 2012-February 2013, and phase 2 testing occurred during January 2014.

it matches the query; it is calculated as the number of true positives divided by the total number of records that match the query.[8]

Second, we define two metrics that describe the performance of queries. We note that query performance is a function of several variables, including query type and complexity (i.e., what the SQL `where` clause says), result set size, and the amount of information about matching records to return to the querier (i.e., what the SQL `select` clause says). As a result, we measure query performance for several different settings of these variables. Choosing the proper set of variables is made more complicated by the fact that the decision depends on the SPAR team: as an example, BLIND SEER's performance on conjunction queries is a function of the size of the most selective clause whereas ESPADA's performance on the same queries is a function of the size of the first clause.

Concretely, *query latency* measures the amount of time (in seconds) required to answer a single query. The measurement starts when all previous queries have been answered and the test harness is ready to send the next query command. Measurement ends when the test harness receives the last byte of decrypted result from the querier.

Additionally, *query throughput* measures the number of queries the prototype implementation can, on average, answer in one second. In order to test query throughput, the test harness will send a test script containing n queries to the SPAR implementation as fast as possible (i.e., the test harness will send another command as soon as the querier indicates that it is ready to accept one). The query throughput is computed by dividing n by the number of seconds that elapsed between the test harness scheduling the first command and the querier receiving all of the decrypted query results (in any order).

Note that these two metrics start measuring time when the test harness is ready to send a command to the SPAR implementation. However, the test harness will not actually submit a query until the SPAR implementation reports that it is ready to receive one. As a result, if a SPAR implementation requires a delay before accepting a new query (e.g., if it first performs some pre-computing work that is independent of the query), then our timer may begin before the test harness actually sends a query.

Policy tests This section describes the process by which we evaluated query authorization policy mechanisms. As before, we wish to measure accuracy and performance. Because policies are expected to be relatively static, the amount of time required to initialize a policy is uninteresting. Instead, our tests load different policies into a SPAR implementation and then measure their effect on the correctness and performance of subsequent queries in 'latency' mode.

The set of queries used in this test is relatively small, as the test is purely meant to confirm that all policy functionality is working as expected. The critical feature of our query set is that, for each policy being tested, some of the queries are compliant with the policy and some are not.

Because each technology supports a substantially different class of policies (see Section

[8]The "ground truth" for the records that match a query is recorded by our data & query generator, and it is verified later by our baseline database software. Both software packages are described in Section 3.1.2.

5.4 for details), our tests are specific to each technology. We ensured that our test policies collectively utilize all of the interesting features supported by each technology, such as white/blacklists on specific fields or values, checking that a property holds for each clause of a disjunctive normal form (DNF) formula, and so forth.

We calculate two metrics from our tests. *Policy precision* is the probability that a query does not conform to the policy given that it was rejected. It is calculated as the number of true positives divided by the sum of true and false positives (as defined in Table 4). *Policy recall* is the probability that a query is rejected given that it does not conform to the policy. It is calculated as the number of true positives divided by the sum of true positives and false negatives.

We also measure query latencies. Because our policy tests execute a relatively small number of queries, we do not anticipate that the query latency information gathered here will be as expansive as the information collected in the query-specific tests above. Instead, our intent here is to observe the *difference* in running time for a query when a policy is active and when it is not. We analyze this overhead as a function of policy complexity.

Modification and verification tests Finally, Lincoln evaluated the SPAR implementations' ability to handle database modifications and verification checks by the data owner. All types of modifications (insertions, updates, and deletions) were tested for correctness and performance in both 'latency' and 'throughput' modes. Special query sets were run before and after the modification in order to detect the change to the database. Verification tests were executed similarly. Finally, we performed atomicity tests to ensure that database modifications were performed reliably.

In the interest of brevity, this this report omits the results of modification and verification testing during the research program. Modification test results were mostly subsumed and improved by the results of SPAR Pilot testing, and verification tests results were straightforward and uninteresting. We refer interested readers to Lincoln's reports at the end of each phase for details on modification and verification testing.

3.1.2 Test Methodology

All tests during the SPAR research program were conducted at MIT Lincoln Laboratory. The tests ran on hardware that we architected specifically for SPAR testing, and they were instrumented using software that we prepared beforehand. MIT LL also produced the synthetic databases and queries used during testing. This section describes MIT LL's test procedure and the rationale behind our decisions. We refer interested readers to our published materials [20, 62] for more details.

Test Environment Our test environment incorporated commodity hardware that was specifically chosen to model a database setup on a local area network. It purposely has powerful computational and networking specs to be amenable to the resource-hungry SPAR technologies. Concretely, our environment contained five Dell PowerEdge R710 servers with:

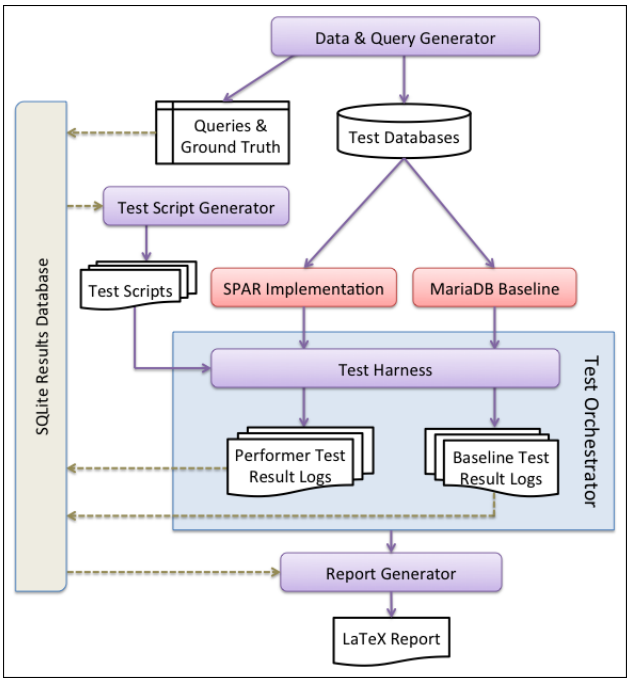


Figure 4: SPAR research program's test framework (showing one implementation). This flow diagram denotes the sequence of execution of the various components of MIT LL's test framework along with the data products produced by each component.

- Two hex-core Intel Xeon X5650 processors (12 cores total) that could quickly perform the heavy cryptographic operations in the SPAR technologies.
- 96 GB of RAM so that the encrypted index mechanisms could fit entirely inside RAM and thus be faster to search.
- Two separate gigabit ethernet adaptors. One network was used exclusively by the test team to coordinate the testing and gather data, and the other one was used exclusively by the SPAR implementations. The isolation ensured that our test harness did not interfere with their network latency.
- Hardware RAID controllers to connect to data arrays of up to 50 TB in a RAID5 configuration. Although the largest database we tested was 10 TB in size, extra storage was required because (1) encrypted data can take up more space than its corresponding plaintext data and (2) temporary disk space is needed during index generation.

All servers ran the Ubuntu 12.04LTS operating system.

Test Framework Our assessment framework for TA-1 is shown in Figure 4. We developed the framework based upon three primary guidelines. First, we wanted to capture comprehensive measurements for query latency and throughput at the 10 TB scale with minimal

overhead. Second, we needed to consider the unique difficulties of testing privacy-preserving systems; for example, encrypted communications between parties precluded the test framework from performing traffic inspection. Third, we wanted to automate as much of the testing procedure as possible in order to maximize human time available. To achieve these goals, our framework contains the following components:

- Our *data generator* built artificial but realistic-looking data corresponding to people's names, addresses, birthdays, etc. based upon data from the US Census Bureau [59, 60] and Project Gutenberg [29] in a repeatable fashion. Additionally, our *query generator* designed queries that were well-distributed among several variables of interest, such as the number of records returned. These queries were partitioned into *test scripts* based on the genre of query executed (keyword, boolean, range, etc.) and the expected number of results for each query. See Section 3.1.3 and Appendix C for details.

- In order to provide an "off-the-shelf" non-privacy-preserving database as a *baseline* for comparison, we installed the MySQL (in phase 1) and MariaDB (in phase 2) open-source database systems in our test environment. Record caching was disabled on the baseline systems to provide for a better comparison.

- Our *test harness* actuated and monitored the SPAR implementation and baseline systems. It used the previously-generated test scripts to coordinate the execution of queries and database updates. Detailed, timestamped outputs were saved in test logs. We optimized the test harness' performance so that it could handle massive string I/O with minimal CPU, RAM, and disk usage.

  The test harness contained two pieces, called `client-harness` and `server-harness`, which operated on the same machines as the data querier and owner software, respectively. The `client-harness` kicked off the test and fed queries to the querier software. The `server-harness` fed modification requests to the owner software. The two harness pieces coordinated their execution by communicating over an isolated network.

- Before each test, our *test orchestrator* automated the procedure of installing the SPAR implementation software and test harness (with the appropriate test script) on each Dell R710 machine. After the harness indicated that a test completed, the orchestrator automatically stopped the implementation software and archived all test logs.

- Our *report generator* automated the post-test process of analyzing the test logs to produce a human-readable report. Indeed, many of the figures in Section 6.1 of this report were automatically generated by our report generator software. It pulled data from a common SQLite database into which the results of prior steps were inserted.

Test Execution One week each was allocated per SPAR implementation for testing. Each SPAR team was provided with remote access to our test environment 1-4 weeks before their test for database ingestion and indexing. Then, a subset of each team traveled to Lincoln for one week of query, modification, and policy testing.

To properly characterize SPAR implementations, our assessment framework executed each query test script twice: in a *latency* mode in which the harness fed one query at a time to the implementation software, and in a *throughput* mode in which the harness sent queries as quickly as possible to measure the degree of parallelization of SPAR implementations. Lincoln also executed test scripts for modification testing during the research program, but we omit those results as they are largely subsumed and improved during SPAR Pilot testing.

Risk Mitigation During the research program, SPAR implementations had a "research-ware" quality (see Section 7) and often failed in unpredictable ways. As such, we employed five mechanisms to increase the likelihood of acquiring useful data during the testing period:

1. Provided all SPAR teams with remote access to portions of our test environment for several months.

2. Conducted a formal *risk reduction* activity 1-2 months prior to testing, during which we conducted a minimal test using our test harness in order to reduce the likelihood of errors during the final testing period.

3. Executed a *smoke test script* comprising a small set of queries that span the types of queries that would be executed during the full test, in order to expose problems early in the test week and give teams adequate time to develop patches.

4. Manually cataloged all test events (especially anomalous ones) to help the implementation teams realize when an error occurred and to help the Lincoln test team determine which test results should be discarded.

5. Used the automated report generator during the testing period to visualize and identify anomalies without having to manually search through test artifacts; this sped up the process of diagnosing bugs and configuration issues.

3.1.3 Synthetic Test Datasets

Table 2 shows the 9 different database sizes on which we conducted testing. The single-table databases differ in their number of records and the width of their schema. The precise schemas are detailed in Table 17 in Appendix C. For the discussion in this section, it suffices to state that the schema contains a unique identifier `id` field, several small fields of various data types (dates, enums, integers, and strings). The 'wider' databases with 10^5 byte record sizes also contain variable-length searchable string fields (called `notes`) of up to 10 KB in size and an unsearchable binary blob field (called `fingerprint`) of up to 100 KB in size.

All 9 databases have a common data generation proedure. MIT LL produced synthetic databases consistent with the schemas based upon data from Project Gutenberg [29] for all `notes` fields, random data in the `fingerprint` field, and data from the US Census Bureau [59, 60] for all other fields. Databases with the 10^2 row schema were generated in the same manner with a reduced set of fields populated by the US Census Bureau data.

Approx DB Size (bytes)	# of Records	Approx Record Size (bytes)
100 MB	10^6	10^2
1 GB*	10^4	10^5
10 GB	10^8	10^2
10 GB	10^5	10^5
100 GB	10^6	10^5
100 GB	10^9	10^2
500 GB*	$5 \cdot 10^6$	10^5
1 TB*	10^7	10^5
10 TB	10^8	10^5

Table 2: Database configurations. Rows with an asterisk (*) were only tested in Phase 1. Rows with the smaller 10^2 byte/row schema were only tested in Phase 2.

For repeatability, data generation was parameterized by a single random seed; this allowed SPAR teams to generate identical databases to ours and thus obviated the need to transmit large data sets. Data generation for the largest (10 TB) database took 9 hours to complete on the Dell R710 machines in the test environment.

MIT LL's query generator supported two types of `select` clauses: either it would simply request the unique `id` field (by requesting `select id`) or it would request the entire record contents (a `select *` query). To understand the difference between these two queries, recall that SPAR implementations have small encrypted indices that fit in RAM and large data-stores on disk for unstructured data (mainly the `fingerprint` field). A `select id` query typically can be answered from the index structure alone, whereas a `select *` query requires following pointers from the index structure to the relevant portions of the unstructured data-store. Hence, by collecting performance metrics for both types of `select` queries, MIT LL can compare the performance of the two types of data structures used in SPAR technologies.

Additionally, our query generator built `where` clauses over all fields in this schema except for the `fingerprint` blob. However, all teams requested some limitations on the fields (or combinations of fields, in the case of a conjunction query) that were allowed to be searchable during their tests. As a result, MIT LL customized the query set for each implementation to comply with their limitations. Additionally, the `where` clauses produced by our query generator finely balanced two needs: (1) executing queries of all query types, record set sizes, and all other variables that were deemed to be of interest and (2) calibrating the number of queries so they could reasonably complete in 2-3 days, based upon query-response speeds observed during the risk-reduction period, so that teams could reasonably expect to complete their testing within one week even with some time reserved for debugging.

A concrete description of the full set of queries is presented Appendix C. Intuitively, to balance these two goals, we skewed our query set to contain several queries that were fast (e.g., had a small number of matching records) and a smaller number of slow queries; nevertheless, our distribution was calibrated to ensure that best-fit curves could be appropriately drawn. For example, all queries in our test set were executed in the faster `select`

`id` mode, whereas only some queries were repeated in the slower `select *` mode (cf. the distribution of data points in Figure 9 in Section 6). Additionally, all of the queries were run in latency mode, whereas we only executed some of the queries in throughput mode. Each throughput test script only queries that return between 0 and 100 records so that a strong SPAR implementation would benefit from supporting parallelism between queries and not just parallelism within a query.

3.2 Pilot Demonstration

This section presents the methodology for the SPAR pilot test. The three SPAR implementations were evaluated in two separate environments. The pilot testing focused more on the usability of SPAR technologies. The metrics collected are at a higher level than the research testing. Furthermore, testing in the pilot environment focused on the aspects of that environment that were unique and could not be replicated in synthetic tests. Testing took place in two environments described below:

Risk Reduction Environment The three SPAR implementations were first evaluated in an unclassified cloud environment developed at MIT LL. This environment allowed us to understand the installation and configuration process for each implementation. During June 2015, MIT LL performed a risk reduction test where MIT LL deployed and configuration each SPAR implementation. The implementations were then connected to the MIT LL test infrastructure. Test personnel then issued a few basic queries, performed inserts/deletes, and configured policy enforcement. We call this environment the *risk reduction environment* and describe it in Appendix D. After this, IARPA and MIT LL jointly decided to include all three technologies in the pilot demonstration.

Pilot Environment The pilot demonstration took place in a classified virtualized cloud environment known as commercial cloud services (C2S). This environment used an Amazon web services (AWS) backend. The data and use case are described in the addendum to this document [10].

In this pilot demonstration, MIT LL asked genuine database operators to use and provide feedback on the three SPAR technologies. MIT LL then performed automated testing on each SPAR implementation. To provide more context for understanding the findings in this document, Appendix E presents findings from the previous research program conducted on SPAR technology.

In order to gain a better understanding of the capabilities and limitations of each SPAR technology, the pilot demonstration included each of the following. Time in the pilot environment was limited; some testing was done in the risk reduction environment and is noted below.

- Live operator testing to collect user-sentiment data and operator-created queries

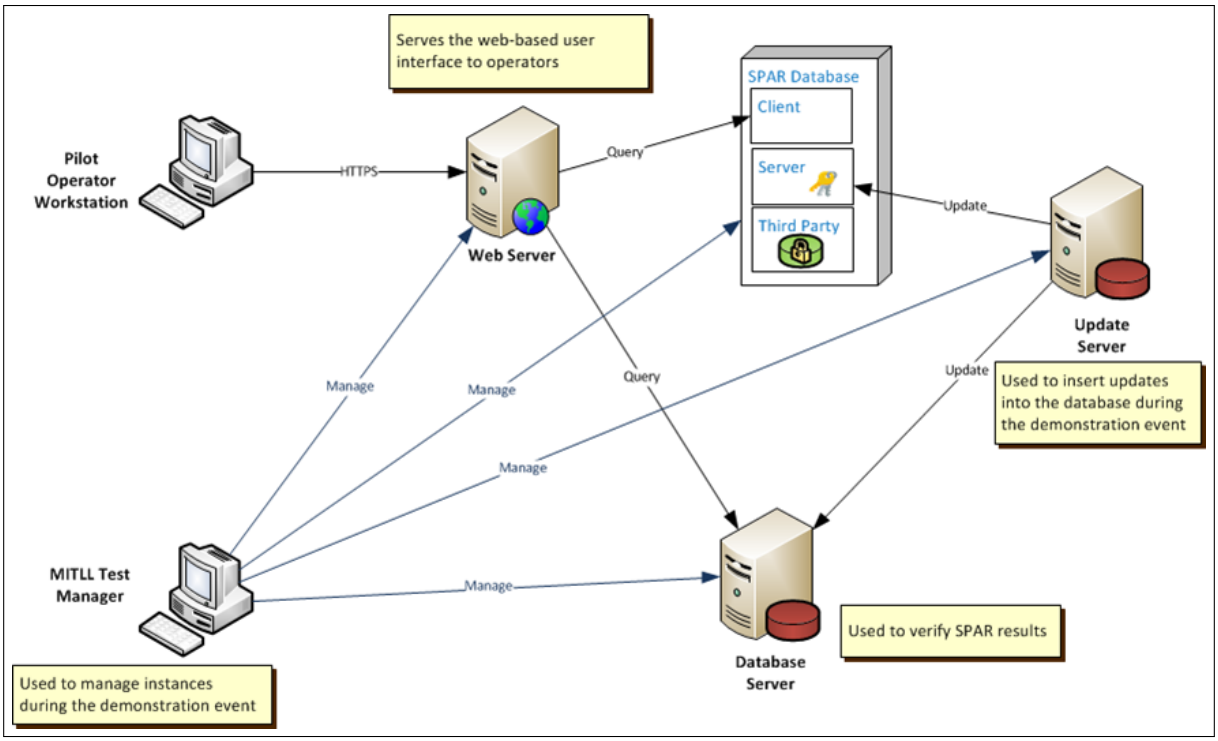


Figure 5: SPAR Pilot test apparatus (showing one SPAR implementation).

- Automated testing on basic database operations – query time, insert time and accuracy, and delete time and accuracy[9]
- Automated testing on query-checking policy[10]

This report also describes MIT LL's experience deploying each SPAR implementation (Section 7). This section is based on MIT LL evaluation personnel's anecdotal experience in both environments.

3.2.1 Test Apparatus and Management

This subsection describes the environment in which the SPAR software was tested in order to provide context for understanding the testing results.

The overall architecture for the test environment can be found in Figure 5 which illustrates the system concept for a single-implementation system in the SPAR pilot test environment. The SPAR database, a web server, a database server, and an update server were hosted in a cloud.[11] These systems were managed by the MIT LL test manager, located at MIT LL,

[9]Delete timing and accuracy were performed in the risk reduction environment.

[10]Cryptographic policy enforcement was tested in the risk reduction environment.

[11]During the risk reduction testing this was an Openstack based cloud environment as described in Appendix D. During the pilot demonstration, the cloud was a classified Amazon Web Services environment called Commercial Cloud Services (C2S).

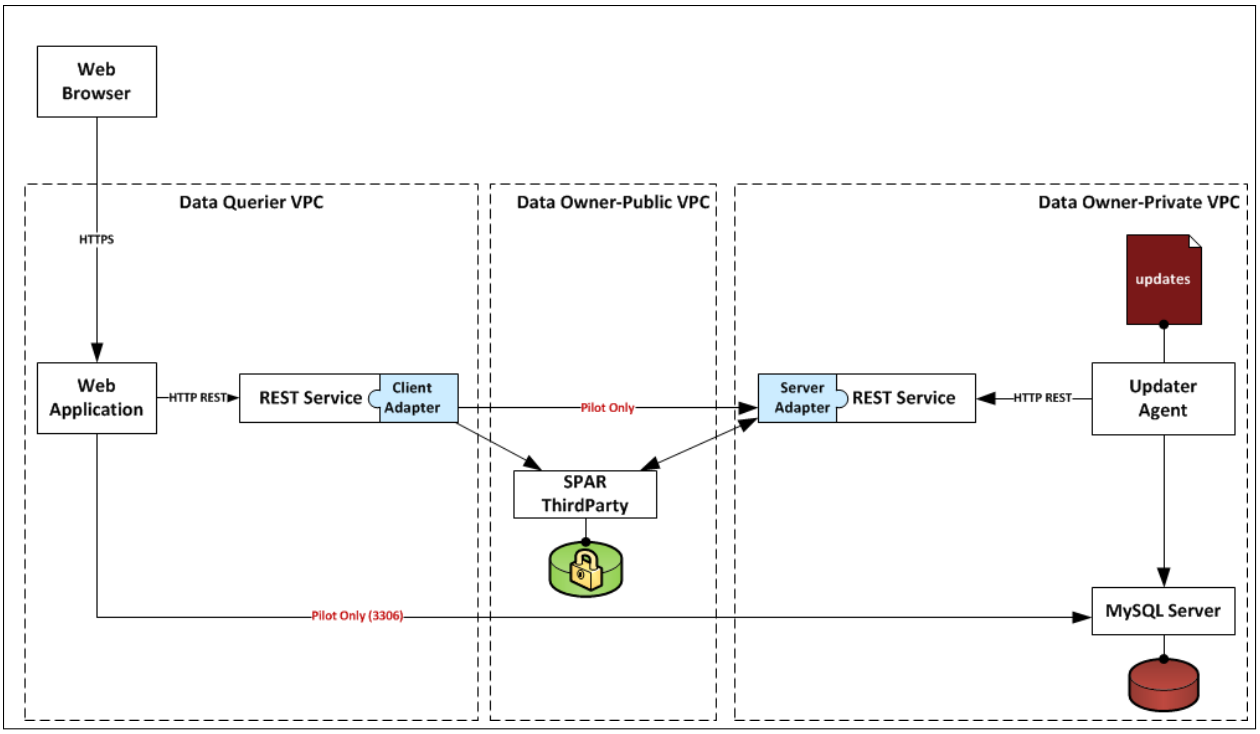


Figure 6: SPAR Pilot Process View (showing one implementation). This illustrates the runtime connectivity between the processes hosted on each systems. The dashed lines depict trust boundaries between organizations. An arrow from process A to process B illustrates that process A initiates the communication with process B, which is listening for an incoming connection. An arrow in both directions indicates multiple channels of communications between the processes. The two connections marked "Pilot Only" would not be required in a production version of the system. Depending on the use-case, the MySQL server could also be removed in a production system.

and queries executed by a operator workstation, located at the Data Querier's organization. Not shown in the figure is the MySQL instance run in parallel to the SPAR database in the cloud which served as a baseline for the system.[12]

Figure 6 illustrates the runtime connectivity between different processes in the overall architecture of the pilot test environment. Live operators interacted with SPAR technology through a web application that included a visual query builder, shown in Figure 7, that queried the underlying SPAR database. The interface was consistent regardless of the back-end SPAR implementation, providing continuity to the operator's experience during testing. Once the query was built, the web server submitted the query to the SPAR implementation and MySQL baseline REST services. All communication with the SPAR implementations, both client and server, occurred through a REST interface implemented by the MIT LL

[12]The MySQL system was configured with single column indices for each searchable field in the pilot schema.

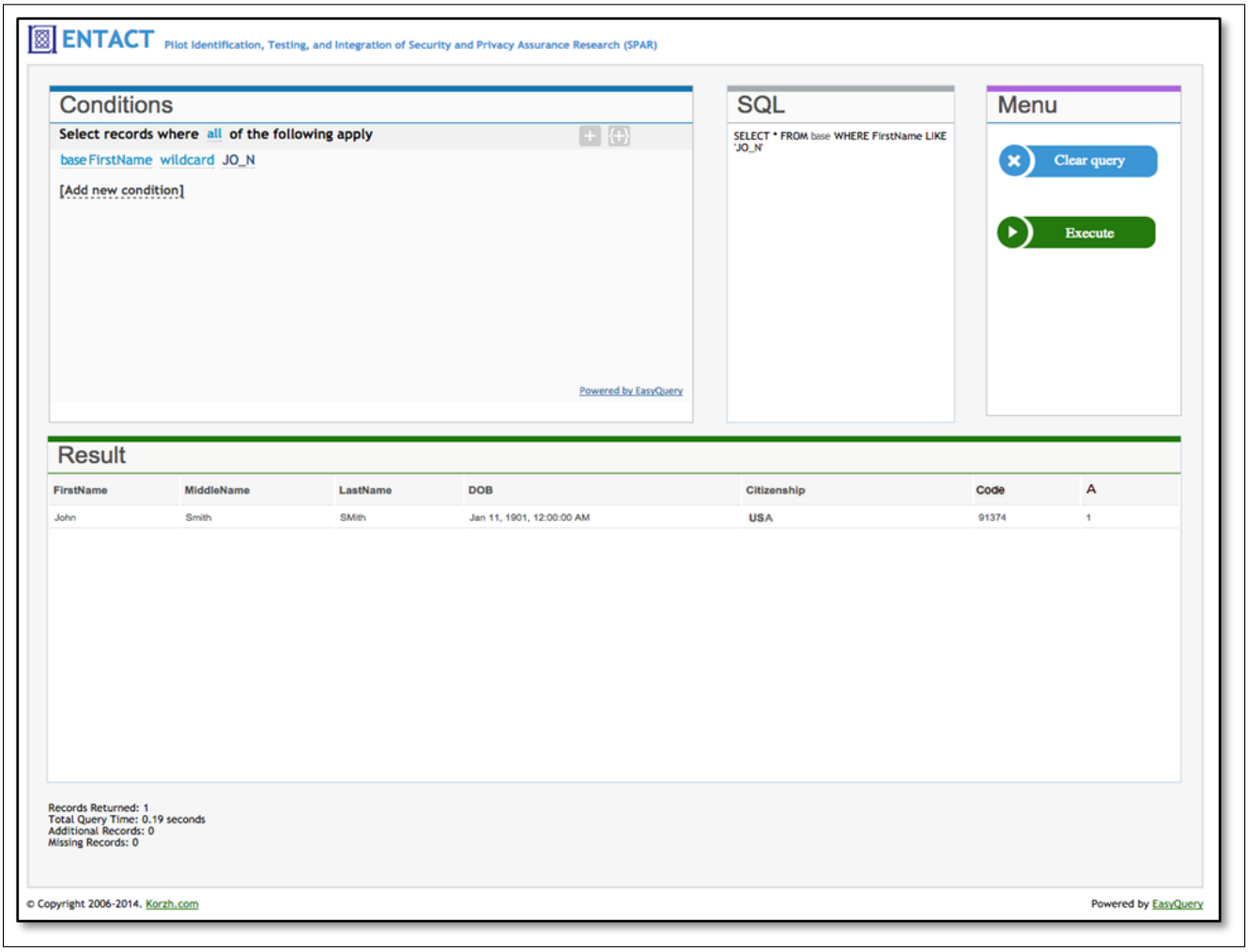


Figure 7: SPAR Pilot Web Interface (showing visual query builder)

team. Query requests, either from the web service, or submitted during automated testing, passed through the REST interface to the SPAR implementation's client adapter library. Similarly, updates were handled through a server REST service. The use of a RESTful service for both the client and the server allowed the MIT LL team to easily integrate different interfaces with the SPAR implementations.

Prior to the pilot period, MIT LL provided SPAR teams with a standard API for client and server adapter libraries modeled after the MySQL API.[13] The SPAR teams wrote adapter libraries to this API in either C or C++ which where then integrated into the REST services. The use of a programmatic API improved functionality specification, testing, integration, and installation. An important part of this API was the ability for implementations to report queries as unsupported. This was the desired behavior if a query could not be parsed or implementation could not answer the query.

After receiving the query, the SPAR client adapter managed the interaction with the SPAR server and the SPAR third party to answer the query and ultimately returned the results. Upon the receipt of the results, the web server compared the results obtained from the SPAR implementation with the results from MySQL to verify the correctness of the results and report differences. Unlike the SPAR Research testing inaccurate results are not separated by missing or extra results. Queries are listed are accurate or inaccurate. This information and timing information for each of the systems were reported to the user in the web browser along with the results. Results, including the query, number of results, timing and correctness information were logged to the web server machine.

Query performance timing in this document is the time when the query was issued at the web server to when the last result was returned from the RESTful service. This time contains multiple network hops that did not exist in the SPAR research program. Furthermore, this was over a production network that contained other traffic. During the pilot all queries were issued in `select *` mode, retrieving the entire record set.

After operator testing, an update server served updates simultaneously to SPAR and the MySQL baseline. Again, this communication happened through the RESTful service to the SPAR server adapter libraries. Update timing is measured from the time of the first update command until the last update has been processed. This process also contains multiple network hops that were not present in the SPAR research program.

3.2.2 Preparation and Changes to the Original Testing Plan

The methodology used for the pilot demonstration deviated significantly from the original plan. The original plan for the pilot demonstration is found in Appendix **??**. That plan was developed under the expectation that all configuration and ingest tasks for each SPAR implementation would be completed prior to the beginning of the pilot demonstration.

Data-formatting issues caused delays. The SPAR implementations work with a single database table. However, MIT LL received the pilot demonstration data as XML. Parsing these XML data into a database table took several days. Further, there were delays caused

[13] As described in the previous subsection, during SPAR Research testing, MIT LL's test harness worked through standard I/O channels.

by documentation gaps, configuration problems, and ingest problems. While MIT LL had originally included time for troubleshooting the systems before the pilot demonstration, the actual time required to overcome these challenges greatly exceeded expectations. As SPAR implementations mature, they need to be more flexible in handling different data formats. We discuss this further in Section 7.

Only a single SPAR implementation was deployed when the pilot demonstration began. With timely support from SPAR teams, configuration and ingest on the other two SPAR implementations was completed during the initial training phase of operator testing. This allowed the operator testing itself to begin on schedule. Importantly, all operators trained using the same SPAR implementation. However, these delays impacted the preparations for database updates that were originally scheduled to occur in the operator-testing schedule. In addition, one of the organizations experienced network connectivity problems during the pilot demonstration. These delays made a reduction in the scope of the operator testing unavoidable. MIT LL made a determination that the impact of updates and deletes on query performance could be measured with automated testing after operator testing had concluded, and decided to defer this testing until after the operator testing was completed.

An unfortunate consequence of transforming the pilot data in a small time period is that there were erroneous data-row duplicates in the database table. MIT LL had planned to extract 10 million unique records for the operator study, but instead extracted approximately 1.7 million unique records and 8.3 million duplicates. There was not time to remove these duplicates before live operator testing was scheduled to begin. As a result, operators often encountered multiple identical records. This made it difficult for operators to know if they received the exact data they had requested. This also decreased the overall entropy of the database. The performance of some SPAR implementations depends on data entropy; we return to this issue when discussing implementation performance. We stress this problem was due to the MIT LL transformation procedure.

3.2.3 Operator Testing

The pilot demonstration included live operators interacting with each of the three SPAR technologies. There were ten operators, including a mix of technical and non-technical, with six men and four women. While the operators were aware that they were evaluating three different systems, these systems were only identified as A, B, and C. MIT LL provided all operators with initial training on the user interface and operators were not given any information about the technical limitations of any of the technologies. All operator testing was conducted on a static dataset of 10 million records.

MIT LL attempted to follow a latin square design whenever possible, with each operator using a different SPAR implementation each day. However, operator availability made this impossible to follow. All operators received the training before starting the other steps in the testing protocol. Most operators only completed a subset of the overall steps in the testing protocol. However, the operator testing still provided valuable data: both actual user sentiment on using each SPAR technology as well as real operator-generated queries that would be used during subsequent automated testing. Because different operators were

participating in different steps of the protocol at different times, the data remained static through the entire operator-testing phase.

Operators participated in up to three different sessions. Due to operator availability, operators often participated in these sessions at different times. Operators received training on using SPAR and its user interface. Then operators were asked to execute a pre-scripted set of queries. Finally, operators were asked to create whatever queries they thought appropriate. This final step provided a trove of operator-generated queries for use in subsequent automated testing. After each session, the operators filled out a questionnaire regarding their experience with the technology.

- Session 1. Training: Operators were familiarized with the technology and the user interface. All ten operators participated in this step. Nine of the ten operators used the Stealth SPAR implementation for training because the other two implementations were not yet ready.

- Sessions 2-4. Scripted: Operators were assigned a different implementation each day and asked to complete a test script of queries specified by MIT LL. Operators had a thirty-minute session with each implementation. This allowed MIT LL to ensure that operators conducted a broad set of queries. MIT LL collected data from seven to nine different operators for each SPAR implementation.

- Sessions 5-7. Exploratory: Operators were assigned a different implementation each day and given an opportunity to execute their normal workflow and enter any queries they wanted. If operators were unable to execute a desired query, MIT LL provided coaching on limitations and guidance on how to structure the query to obtain the desired results. All of the queries entered by operators were logged during the pilot demonstration. The list of queries was then de-duplicated and used to create a query corpus for use in automated performance testing after the pilot demonstration. MIT LL collected data from three to five different operators for each SPAR implementation.

Lack of Control Sessions: Often, when evaluating usability of a new technology, users interact with the new technology and a baseline technology during different sessions. This allows the test team to separates effects due to the underlying technology and effects due to the overall experiment design. MIT LL hoped to use this methodology during the pilot demonstration. However, due to sensitivity of the data used in pilot, operators were not able to directly interact with the baseline MySQL database. This means that operators' responses are influenced by both the experiment design (including the user interface) and the underlying technology. We revisit this issue in Section 8.

3.2.4 Automated Testing for Timing, Insert, and Delete

MIT LL used automated testing to measure query-response time as well as how each SPAR implementation handled both insert and delete operation. As discussed above, the set of all queries used by the live operators became the query corpus used in this automated testing.

In order to measure timing, for each implementation, each query in the query corpus was submitted for processing. The round-trip time between query submission and the return of results was logged. Additionally, the returned results were compared against a MySQL baseline for accuracy and timing.

After the first round of automated performance testing was completed, staged database inserts were performed. Three insert steps were performed on all implementations: adding 10 thousand, 100 thousand, and then 500 thousand records.[14] The time required to complete these inserts was measured. The reported times are from the issuance of the first insert statement to the time when the last insert statement was processed by the implementation. Then a second round of query-time performance testing was conducted to measure the impact of the inserts on subsequent system speed and accuracy. SPAR technologies have very difficult approaches to handling inserted records and this affected some testing (discussion in Section 6.2).

3.2.5 Automated Testing for Query-Check Enforcement

A feature of SPAR technology is allowing the data owner to specify rules that the data querier must follow in constructing its queries. This is called a *query check*. The way in which a query check should evaluate a prospective query is specified as a set of *query check rules*. For a given set of query check rules, a given query should be either accepted and evaluated (a *valid* query) or it should be rejected and not evaluated (an *invalid* query).

Valid/Invalid	Whether a given query should be accepted or rejected
Accepted/Rejected	How a SPAR implementation responds to a particular query

Table 3: Terminology used to describe a query check.

This analysis focused on whether and how each SPAR implementation was able to enforce query check rules. The same 909 queries were issued against each SPAR implementation.

True positive	Occurs when an invalid query is correctly rejected by a query check
True negative	Occurs when a valid query is correctly accepted by a query check
False positive	Occurs when a valid query should be accepted but is rejected
False negative	Occurs when an invalid query should be rejected but is accepted

Table 4: Terminology used to describe the results of a query check.

To limit risks due to implementation failures and to allow discussion with SPAR teams, the query check evaluation was performed in the risk reduction environment. As described in Appendix D, the risk reduction data was designed to resemble insurance data but with

[14] A 1 million record insert was also performed on the Stealth implementation.

similarities to the genuine data used in operator testing. The BLIND SEER and Stealth systems were evaluated on a 10 million record data set. The ESPADA implementation was evaluated on a much smaller 100,000 record data set. This is because the ESPADA system had problems working with the file system in the risk reduction environment. These problems necessitated the use of a smaller data set. We stress these problems did not exist in the pilot environment where all systems were evaluated on the same 10 million record starting dataset.

3.2.6 Tested Query Check Rules

MIT LL generated these query check rules based on the rules used in the pilot study [10]. Each SPAR implementation has its own language for specifying query check rules. MIT LL implemented these rules as accurately as possible in each implementation's language.

1. A single Boolean field (`A`) must be present and be 0. A prospective query could be rejected if field `A` is missing, or is not set to be 0.

2. A date range (`issuanceDate`) must be between a specified start and end date (3/1/2015 to 6/1/2015).

3. The date range for `issuanceDate` must be less than 94 days.

For the system's policy enforcement to be usable, it must also accept queries which satisfy all of these rules. We call accepting valid queries *Rule 0*. We now evaluate different aspects of the SPAR technologies.

4 Security

In this section, we detail the cryptographic assurance guarantees provided by the SPAR database technology. This section assumes knowledge of the high-level approaches for each technology described in Section 2.

First, we describe the SPAR program's security model, which uses terminology and models from the academic literature but is motivated by threats in government use cases. We stress that SPAR's threat model focuses on cryptographic protections and as such does not provide a complete system guarantee; for instance, it provides no protections against malware infecting any of the machines used, nor does it require that the SPAR implementations themselves are free of software vulnerabilities. MIT LL has not evaluated the security of SPAR software implementations. Any full deployment of SPAR technology would require a security evaluation of the entire system including the software prototypes. SPAR technology should undergo a through security evaluation before being operationally deployed, as recommended in Section 1.7.

Second, we define and describe the SPAR BAA's 12 assurance requirements with regards to the protection of queries, database contents, multiple clients, and policies. Concurrent with these definitions, we summarize the SPAR technologies' common strengths and weaknesses in meeting these guarantees.

Third, we summarize the extent to which each technology meets the BAA's assurance guarantees and the cryptographic assumptions required to do so. Then, we highlight each individual technology's unique capabilities. Finally, we note each technology's shortcomings and assess the likelihood that they can easily be addressed without a substantial overhaul to the technology.

4.1 Threat Model

Recall that SPAR's database management systems involve three parties: a data owner, a data querier who wishes to query the data, and a database server that holds the owner's data in encrypted form and responds to the querier's queries. The BAA and Rules of Engagement require that SPAR technologies meet certain assurance requirements against all three parties [22, 44, 55].

Before explaining the SPAR requirements, we first describe a few different types of adversarial behavior that are commonly considered in cryptographic protocols.

Semi-honest: A party is said to be semi-honest, or "honest-but-curious," if it follows the cryptographic protocol honestly (in particular, performing the computations asked of it and transmitting messages in the proper formats), but it attempts to glean additional information from messages that it observes. Semi-honest parties do not eavesdrop on messages between other parties.

Malicious: A malicious party does not need to follow the prescribed protocol; it can interact with the other parties in any way that it sees fit to maximize its chances of learning unauthorized information.

Collusion: Parties are said to collude if they communicate outside of the prescribed protocol in order to share data they have received or to execute a malicious attack jointly.

SPAR teams proved the security of their cryptographic protocols against all parties individually acting in a semi-honest manner. Additionally, technologies that supported multiple queriers were required to show that multiple colluding clients (sharing all of their communications) could not learn any additional information. Finally, SPAR Phase 2 implementation teams proved that their protocols withstand malicious queriers who attempt to deceive the owner and database server into answering a query that should be rejected by the owner's policy.

Additionally, SPAR teams were required to secure their protocols against a semi-honest or malicious outsider who sees and controls all network communications (but does not collude with any participant in the protocol). This network adversary sees all traffic and is allowed to insert, remove, maul, or replay any communication to any party.

No other types of security were provided. In particular, SPAR's threat model does not include a database server or data owner who act maliciously or who collude with each other or with any queriers. Indeed, if the database server colludes with either the owner or querier, this colluding pair can learn a lot of information about the querier's queries or owner's data (respectively). Additionally, the three legitimate parties are assumed only to see their traffic (and assumed not to collude with the network adversary).

4.2 Assurance Requirements

In this section, we define and explain the SPAR BAA's 12 assurance requirements [22, Table 3]. We partition the assurance requirements into four categories that protect queries, database contents, multiple clients, and policies. Alongside the descriptions, we summarize the SPAR technologies' common strengths and weaknesses toward achieving these security requirements. We defer a discussion of the differences between technologies to Section 4.3 and Table 5.

Admittedly, the remarks in this this section skew toward the SPAR technologies' common deficiencies; i.e., the types of information that the technologies reveal or *leak* in violation of the assurance requirements. As such, we stress upfront that the SPAR teams did very well in meeting (and in some cases exceeding) the requirements overall; these deficiencies should be viewed as imperfections toward an incredibly high goal that that program set out to accomplish.

Queries. The first three assurance requirements protect the contents of the queries from a semi-honest data owner and database server.

TA1-A1. The data owner and database server should not be able to "learn any information about an individual query with the exception of a minimal amount of information about query access patterns." In practice, SPAR technologies' ability to meet this guarantee varied significantly based on the type of query performed. For equality, keyword,

stemming queries, and disjunctions, the owner and server learn only the number of records returned and (for some technologies) the attribute field being searched.

However, the SPAR technologies leak more information on conjunction, range, wildcard, subsequence, threshold, and ranking queries. Examples of leakage include the number of records that would be returned by each individual clause in a conjunction query or the number of characters in a wildcard or subsequence query.

TA1-A2. The owner and database server should not be able to "learn any information about the identity or content of any records returned." Most SPAR technologies met this requirement exactly as long as the querier can maintain a cache of all records retrieved so as not to make a duplicate request for a record (otherwise, the database server would learn the number of times each record had been accessed).

TA1-A3. As stated above, most technologies leaked the number of records returned in response to a query. This requirement imposes a limitation on such leakage: the owner and database server should not be able to distinguish between queries that return 0 records and queries that return 1 record. This requirement is intended for applications where queries are so targeted that they often return no records, and even the existence of a (rare) match is deemed to be sensitive information. Technologies did not meet this requirement fully, although the exact nature of their limitations varies substantially by SPAR technology.

Data. The next three assurance requirements protect the contents of the owner's database against a semi-honest querier and database server.

TA1-A4. During the execution of a query, the database server should not learn the raw contents of any data, and the querier should not "learn any information about records not in the result set of a query." SPAR technologies meet this requirement very well overall, albeit with a few common weaknesses: (1) the length of each record is revealed to the database server and sometimes the querier as well, and (2) conjunction queries reveal the number of records in the database that match some of the search clauses individually, leaking information to the querier about the distribution of data that is not retrieved.

TA1-A5. During and after a database modification, the querier and database server should not learn the contents of any inserted, updated, or deleted data (except if the querier makes an authorized query for this data). The SPAR technologies mostly meet this requirement. However, they typically store modified data in a different data structure than the original database; as a result, the database server (and sometimes also the querier) can tell whether the records returned in response to a query were part of the original database or were inserted later.

TA1-A6. The owner can "efficiently verify the integrity of any record" stored in the database server to ensure that its contents have not been corrupted. The SPAR technologies

exceed this requirement by providing *private* verification of database contents (with the one caveat that the database server might be able to detect when the same record is verified twice).

Multiple queriers. The next two assurance requirements only apply to SPAR technologies that wish to support multiple queriers (a setting that was optional in SPAR). ESPADA is the only SPAR technology that currently supports multiple queriers.

TA1-A7. From a performance point of view, the data owner can efficiently add and remove queriers "without requiring each record to be touched."

TA1-A8. From a privacy point of view, each querier is assured that the database server and other queriers cannot learn whether the owner has given them access to the system.

Policies. The final four assurance requirements ensure that the policy validation mechanism adequately protects the data owner. The protections offered by the policy are intended to ensure that queriers ask targeted queries. They are not intended to restrict access to individual records, but rather to control *how* the records are accessed. While record-based access control is not currently implemented in SPAR, it could be added on top of SPAR technology (if desired) in order to segment access to the data between multiple queriers. Some of the business rules used for the pilot could be easily implemented using record-based access control while other rules needed to look at the structure of the query (see Section 3.2.6).

TA1-A9. The querier is assured that the policy validation mechanism properly allows its valid queries. SPAR technologies meet this requirement up to a very small probability of error.

TA1-A10. The owner is assured that the policy validation mechanism properly rejects all invalid queries. The SPAR technologies do meet this requirement, although we note that some information may leak to the querier on rejected queries, such as the number of results that (clauses in) the query would have returned. We discuss the ability to support the pilot business rules in Section 6.2.

TA1-A11. The owner is assured that even a malicious querier cannot forge the result of the policy validation mechanism. We highlight this requirement as the only one that must hold against a malicious querier. SPAR Phase 2 technologies (BLIND SEER and ESPADA) meet this requirement.

TA1-A12. The owner is assured that other parties cannot distinguish between queries that are rejected by the policy and queries that do not match any records in the database. This requirement limits a querier's ability to "reverse engineer" a policy that the owner intends to keep secret. Like requirement TA1-A3, the value provided by this requirement varies substantially by use case. In general, technologies met this requirement with respect to outsiders but not with respect to the querier or database server.

As a corollary, requirement TA1-A12 implies that the policy validation mechanism must have the ability to keep the policy secret in the first place. Technologies vary significantly in the degree to which they hide the policy. During the pilot demonstration all implementations added a configuration option to make policy failure explicitly visible to the querier. This option can be toggled on/off.

Collectively, these 12 assurance requirements provide guarantees that are similar to, but not quite the same as, those provided by prior academic cryptographic research into secure database searching technologies such as private information retrieval [8], oblivious RAM [17], or searchable symmetric encryption [37].

4.3 Security Evaluation

In this section, we highlight the unique benefits and weaknesses of each of the SPAR technologies. We begin by describing the status of each SPAR technology towards meeting the basic 12 assurance requirements. This section provides only a glimpse into MIT LL's extensive review of each team's proofs and arguments of the security of their technology at the end of each phase of the research program. We refer interested readers to the reports delivered to the government at the end of each phase for more detailed information [49, 51, 54, 56, 57].

The pilot demonstration did not update the security evaluation, other than noting whether the policy technologies were properly being enforced in SPAR implementations. Policy capabilities are discussed in this section, current policy functionality is discussed in Section 5.4. Features added during the pilot demonstration have not been evaluated and may markedly impact each technologies' security. In particular, Stealth added Boolean queries for the pilot demonstration [38]. This is a major new feature whose security has not been evaluated. Any deployment of the Stealth would need a thorough evaluation of their support for Boolean queries.

Additionally, the discussion in this section is based on the technology described by the SPAR teams in their written documents [30–33, 39–42].

4.3.1 Fulfillment of BAA Requirements

Table 5 summarizes the status of SPAR technologies with respect to the 12 security assurance requirements. It describes which of the BAA requirements are satisfied against each party. 'Y' indicates that the requirement is satisfied. 'N' indicates that the requirement is not satisfied. 'P' indicates that the requirement is partially satisfied. '—' indicates that the requirement is not applicable. See MIT LL's SPAR research reports for additional information on partially satisfied requirements [46, 49, 51, 54, 56, 57].

All SPAR technologies reduce the security of their technology to that of other, more standard cryptographic primitives. The teams' written proofs demonstrate that if the assumptions hold, then their technologies are secure. These assumptions are described in Table 6.

		BLIND SEER				ESPADA				Stealth			
	Req.	Q	O	DB	N	Q	O	DB	N	Q	O	DB	N
queries	A1	—	P	P	P	—	Y	P	Y	—	Y	Y	Y
	A2	—	N	Y	Y	—	Y	P	Y	—	Y	Y	Y
	A3	—	Y	P	Y	—	Y	N	N	—	Y	P	P
data	A4	P	—	P	P	Y	—	Y	Y	Y	—	Y	Y
	A5	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	A6	Y	Y	P	Y	Y	Y	Y	Y	Y	Y	P	Y
multi client	A7	N				Y				N			
	A8	N				Y				N			
policies	A9	Y				P				Y			
	A10	Y				Y				Y			
	A11	P				Y				N			
	A12	N	Y	N	P	Y	N	P	Y	Y	N	Y	Y

Table 5: BAA assurance requirements satisfied by each SPAR technology, based on evaluation during SPAR research program. Status of BLIND SEER and ESPADA is shown as of the end of phase 2, status of Stealth is shown as of the end of phase 1. Assurance requirements TA1-A1 through A6 and requirement A12 are validated against an adversarial querier (Q), data owner (O), database server (DB), and network (N). Security is proved with respect to assumptions listed in Table 6.

4.3.2 BLIND SEER

This section briefly highlights some novel properties of the BLIND SEER approach. BLIND SEER's security features were not evaluated in the pilot; this information is drawn from MIT LL's review at the end of phase 2 of the research program [56, Section 3].

The BLIND SEER system offers a few unique capabilities. First, BLIND SEER is the only technology that exceeds requirement A1 by hiding the field being queried from the database server. Second, BLIND SEER partially meets requirement A3 (where some others do not) by restricting the database server from learning whether a query returns 0 or 1 records. Third, BLIND SEER has the most robust policy enforcement language (as discussed in Section 5.4) and the most secure policy enforcement mechanism. Fourth, BLIND SEER's security guarantees are proved in the semi-honest setting for a single querier who submits multiple concurrent queries (perhaps the next-best thing to the multiple querier request).

The main security imperfection of BLIND SEER is that the pattern of traversing through the Bloom filter tree permits the querier, database server, and network to gain information about which intermediate nodes in the Bloom filter tree satisfy the querier's query, which partially violates requirements A1 and A4 (in a perfect scheme, the parties should only learn about matching records). This information is leaked even for queries that fail the policy check, and it can be used to learn statistics of the underlying data. The impact of this leakage grows with the number of queries made; fortunately, BLIND SEER provides a

BLIND SEER	1. AES-128 is a secure symmetric-key encryption scheme 2. 1024-bit El Gamal is a secure public-key encryption scheme 3. SHA-1 and SHA-256 are collision resistant and act as random oracles 4. SSLv3 provides message confidentiality and integrity from a malicious network 5. That a custom-built pseudorandom number generator is secure
ESPADA	1. q-Decisional Diffie-Hellman assumption for elliptic curve P224 2. One-More Gap Diffie-Hellman assumption for the elliptic curve P224 3. A custom-designed hash function (mapping onto the elliptic curve P224) acts as a random oracle 4. AES can be used to build a pseudorandom function, authenticated encryption scheme, and pseudorandom permutation family 5. SHA2-256 can be used to build a pseudorandom function and acts as a random oracle 6. The Linear Diffie-Hellman assumption holds in a bilinear pairing group 7. IPSec provides message confidentiality and integrity from a malicious network
Stealth	1. AES can instantiate a pseudorandom function a pseudorandom number generator 2. That TLS protects confidentiality and integrity against a malicious network

Table 6: Cryptographic assumptions underlying SPAR technology. If these assumptions hold then SPAR systems satisfy the requirements listed as described in Table 5.

"reset" mechanism to mitigate this snowball effect. It would be possible to reduce this tree search pattern somewhat more in the future, but it is impossible to eliminate this leakage entirely within the framework of BLIND SEER's current technology.

Another security imperfection is the fact that BLIND SEER's query and policy mechanisms both have a small but non-zero probability of one-sided errors; this imperfection is unavoidable as it is a weakness of the Bloom filters used extensively in their technology. A final security imperfection is that BLIND SEER's search mechanism reveals the number of records returned by each query to the data owner. This leakage is easily preventable if desired, and it is currently in place to give the data owner rate-limiting protection against a rogue database server returns too many records to the querier.

4.3.3 ESPADA

This section briefly highlights some novel aspects of the ESPADA approach. ESPADA's security features were not evaluated in the pilot, this information is drawn from MIT LL's review at the end of phase 2 of the research program [57, Section 3].

ESPADA provides several unique features. First, it supports multiple queriers; as such, it is the only technology that meets assurance requirements A7 and A8. Second, ESPADA is the only technology that prevents queriers from distinguishing whether a returned record came from the data owner's original database or whether it was inserted/updated later.

Most crucially though, ESPADA is uniquely able to withstand malicious entities (modulo denial of service-style attacks). All security guarantees hold against a malicious querier, or even a colluding set of multiple queriers. Moreover, security holds against a malicious data owner as long as she acts honestly when generating the encrypted database. Additionally, security holds against a malicious database server, as long as it does not maliciously omit or modify results from the final query result. ESPADA's strong security guarantees extend to a collusion between the querier and database server, who jointly cannot compromise the privacy of the owner's data.

ESPADA's biggest security imperfection is that its searching mechanism only computes the results of 2-term conjunctions securely; conjunctions of larger sizes are then computed in the clear. Concretely, on a query of the form $x_1 \wedge x_2 \wedge x_3$, the database server learns the unique identifiers of all records matching $x_1 \wedge x_2$ and $x_1 \wedge x_3$. The database server also learns the number of records matching the first term x_1 alone (but not their unique identifiers). These weaknesses are fundamental limitations of ESPADA's current approach. A consequence of these limitations is that the leakage to the database server during a query is highly dependent upon the choice of the first term in a query; so, the querier must prepare their query appropriately to minimize this leakage.

Other security imperfections include the following: ESPADA makes no claims about the security of modifications against a malicious network, it leaks sizes of a range queries to the data owner (but not their values), its policy mechanism and some query types have a negligible probability of one-sided error, no attempt is made to hide if a search returns 0 or 1 record, and the database server can sometimes distinguish between 0 records returned and policy failure. Many of these imperfections can be mitigated or eliminated if desired.

4.3.4 Stealth

This section briefly highlights some novel aspects of the Stealth approach. Because Stealth's security features were not evaluated in the pilot, this information is drawn from MIT LL's Phase 1 evaluation [54, Section 3]. During Phase 1, Stealth's technology tended to leak less information (i.e., provide better privacy) than its counterparts. However, this benefit was partially due to the fact that Stealth lacked support for conjunction queries, which were the main source of leakage for other SPAR technologies.

Stealth's biggest unique feature is its ability to hide all data statistics from the database server, including whether records are accessed multiple times by the querier. Stealth demonstrated this security guarantee in the research program for single field searches. While Stealth's approach on Boolean queries in the pilot has not been evaluated by MIT LL (see [38]), Stealth claims that their approach leaks only the number of records that match each searched term individually.

Another advantage of Stealth's technology is that its policy mechanism, backed by secure multi-party computation, provides strong security. Unfortunately, Stealth's implementation does not seem to capitalize on the generality provided by MPC; instead, it enforces the smallest policy language of the 3 technologies (see Section 5.4).

Stealth's main weakness is that it is the only SPAR technology that does not prevent a malicious querier from forging the policy check. However, this weakness is mainly due to their participation only in Phase 1 of the program, as requirement A11 was strengthened in Phase 2 to require resistance against malicious queriers. A smaller weakness is that the data owner learns the number of records returned to the querier. We believe that these weaknesses are not inherent to Stealth's technology.

5 Functionality

The goal of SPAR technology is to allow analysts to perform targeted searches over large data sets. SPAR ideally enables analysts to find the proverbial needle in a haystack. SPAR technology is not designed to handle statistical queries or big data analytics.

Analysts perform complicated and unpredictable searches based on outside knowledge and results from prior queries. Ideally, SPAR technology should allow analysts to search in the same way they interact with an unprotected database. SPAR technologies were required to support two high level operations, searching for records and modifying the database (deleting or updating current records and inserting new records). These two operations represent a meaningful subset of query languages but are not complete. This section describes four aspects of SPAR functionality, 1) the types of queries supported by SPAR technologies 2) the correctness of results during both the research program and the pilot demonstration 3) the modification functionality supported by each SPAR technology and 4) the expressiveness of the query check policy language.

5.1 Types of Searches

The SPAR BAA describes eleven query types [22]. These query types were designed to allow analysts a flexible way to restrict large data sets to records of interest. SPAR technologies needed to support a subset of these query types in additional to supporting exact match queries on a single field. Query types are summarized in Figure 8.

SPAR teams were required to identify four of the following query types in Phase 1, and seven in Phase 2, in addition to single-field equality searches. In the pilot demonstration teams were required to support P1, P2, P6, and P7. The pilot demonstration requirements were based on discussion with government stakeholders. Table 7 describes the current search functionality of SPAR implementations. Query types added during the pilot are designed as (Pilot). This distinction is important because MIT LL has not verified the security of these features. Of particular note is Stealth's addition of Boolean queries. This query type was difficult to achieve in the SPAR research program and resulted in security imperfections (see Section 4 for description of security imperfections).

SPAR technologies satisfy the spirit of these query types. However, there are several important caveats:

Boolean ESPADA's performance improves dramatically when queries begin with a conjunction. In their documentation this is known as ESPADA normal form. Stealth supports the AND, OR operations but not negations.

Wildcard Teams were required to support a single wildcard character in any location of the string. BLIND SEER and Stealth support multiple wildcard terms. ESPADA also supports multiple wildcard terms but restricts the location of the wildcard character after the first term.

TA1-P1 Boolean query expressions (including at least three conjunctions).

TA1-P2 Range queries and inequalities for integer numeric, date/time, and at least one other non-numeric data type.

TA1-P3 Free keyword searches without a pre-defined dictionary.

TA1-P4 Matching of keyword variants/keyword stemming.

TA1-P5 Matching of keywords "close" to a specified value, according to at least two definitions of close, e.g., various definitions of edit distance or canonicalization.

TA1-P6 Matching of values with wildcards.

TA1-P7 Matching of values with a specified subsequence.

TA1-P8 m-of-n conjunctions, in which a record is included in the results set if at least m out of n attributes match the queried values.

TA1-P9 Ranking of results, with the rankings learned only by the client receiving the results. Ranking must be with respect to a similarity measure such as edit distance, or m-of-n exact matches.

TA1-P10 Searching relational databases—multiple tables linked by key fields.

TA1-P11 Searching non-tabular data structures, e.g., searching within the content but not the markup of an XML document tree without prior extraction and indexing of the content nodes.

Figure 8: SPAR Search Query Types [22].

Query Type	BLIND SEER	ESPADA	Stealth
P1 - Boolean	Y	Y	P
P2 - Range & Inequality	Y	Y	Y
P3 - Keyword Search	Y	Y	Y
P4 - Stemming	Y	Y	Y
P5 - Proximity Search			
P6 - Wildcard	P	Y	P
P7 - Subsequence	P	Y	Y
P8 - m-of-n	Y	Y	
P9 - ranking results	Y		
P10 - joins			
P11 - XML	Y		

Table 7: Summary of current state of SPAR search functionality. Satisfaction of BAA requirement during research program listed as Y, features added during the pilot are marked as P. Security has not been evaluated for these features.

Subsequence All technologies are limited in their handling of subsequences. First, searches must be for a single subsequence. Multiple substrings must be specified as different search terms. Second, all implementations require a bound on the length of the substring with performance and resource utilization depending on this bound. BLIND SEER and ESPADA require a bound on the minimum length of the substring while Stealth requires a bound on the maximum length of the substring.

5.2 Correctness of Results

The correctness of results was evaluated in both the SPAR research program and the pilot demonstration. In both tests, correctness was defined as returning the same result set as the unprotected baseline system. However, the metrics collected differ between the two tests. As described in Section 3, precision and recall were collected during the research program, while the pilot demonstration just lists the number of inaccurate results. Also, because the pilot demonstration was a black box test SPAR implementations were given the option to return *not supported* for any query. These queries are listed separately.

5.2.1 Research Program

BLIND SEER During Phase 1 testing in the research program, BLIND SEER's query precision was 0.99999 and their query recall was 0.98564. We identified most of their errors as easily-resolvable software bugs caused by string case sensitivity issues and not being able to handle the value '0' in an integer field.

BLIND SEER's software performed worse in Phase 2, with a query precision of 0.751 and recall of 0.987. We break down BLIND SEER's correctness data by query type in Table 8

Query Type	Subtype	Precision	Recall	Query Count
All		0.751	0.987	5223
EQ		1.0	0.999	3052
P1	and	1.0	0.987	371
P1	or	1.0	0.999	503
P2	greater than	0.186	1.0	156
P2	less than	1.0	1.0	129
P2	range	1.0	1.0	59
P3		1.0	0.911	231
P4		1.0	0.999	95
P8		1.0	0.999	253
P9	alarm words	0.964	1.0	40
P9	ranking	0.201	0.882	218
P11		1.0	1.0	116

Table 8: BLIND SEER's query correctness during Phase 2 of the research effort.

and detail their issues here. The low precision in P2 queries is is only due to 2 queries that returned an overwhelmingly high number of false positives when compared to the expected results; when calculated on a per-query basis, the precision is 0.987. The problems with P9 ranking queries were more generally distributed, with approximately 37% of the queries returning non-matching results; even so, on a per query basis the precision is 0.648 and the recall is 0.836. Finally, 70 of the P9 ranking queries returned records in the incorrect order.

ESPADA During Phase 1 testing in the research program, ESPADA's query precision was 1.0 and their query recall was 0.99967. We were unable to determine the cause of their failures.

ESPADA performed similarly in Phase 2, with a query precision of 0.994 and a query recall of 0.981. Additionally, approximately 0.7% of their query responses contained "bad content" (i.e., they returned the correct value for the unique `id` field, but other fields had incorrect data). Their results are broken up by query type in Table 9. We note that ESPADA's low precision on threshold (P8) queries is due to only 1 query that returned an overwhelmingly high number of false positives.

Stealth During Phase 1 testing in the research program, Stealth's query precision was 0.9906 and their query recall was 0.99524. The incorrect query responses fell into two categories: (1) some false positives that are inherent in the fundamental design of their software and thus difficult to remove, and (2) some false negatives in subsequence (P7) queries for which we do not have any explanation, and may simply have been a software bug.

Type	Subtype	Precision	Recall	Bad Content Fraction	Query Count
All		0.994	0.981	0.00741	45448
EQ		1.0	0.999	$1.59 \cdot 10^{-6}$	24895
P1	and	1.0	1.0	0	2621
P1	or	1.0	1.0	$6.80 \cdot 10^{-8}$	4101
P2	greater than	1.0	1.0	0.000611	240
P2	less than	1.0	1.0	0	28
P2	range	1.0	1.0	0	1895
P3		1.0	1.0	0	2299
P4		1.0	1.0	0	1561
P6		1.0	1.0	0	1811
P7	two-sided	1.0	0.923	0	1084
P7	final	1.0	1.0	0	1160
P7	initial	0.9998	0.481	0	1212
P8		0.128	1.0	0	43

Table 9: ESPADA's query correctness during Phase 2 of the research effort. Bad Content Fraction represents the fraction of records where the correct `id` was returned but the records contents did not match the baseline.

5.2.2 Pilot Demonstration

Queries during the pilot demonstration were not separated by query type. We present overall correctness results for the entire query corpus. A query is labeled as one of three options:

Accurately The query returned the same results as the MySQL baseline.

Inaccurately The query returned results that were different from the MySQL baseline.

Error The SPAR implementation either returned *not supported* for the query or returned an error when answering the query. No results are returned for these queries.

We present a single table listing the overall correctness results of the three systems in Table 10. Stealth handled most queries correctly. Stealth had a small number of 'not supported' errors. However, Stealth does not implement negations and there were very few negations issued. ESPADA answered most queries correctly, but did incorrectly return records for some queries (these issues are detailed in Section F.2). BLIND SEER had significant accuracy issues in the queries it answered. The main issue was that BLIND SEER only knew how to answer substrings of length at least four. Many length three substrings were submitted, but instead of reporting not supported, incomplete results were returned. This issue is described in [21].

Ultimately, all SPAR implementations had some issues, but it is worth noting that all three SPAR implementations did greatly expand their functionality during the short amount

Queries Answered	Accurately	Inaccurately	Error
BLIND SEER	156	17	29
ESPADA	190	5	6
Stealth	199	0	3
MySQL	202	0	0

Table 10: Pilot demonstration query expressivity, number of queries. Accurate queries returned without error and with the correct results. Inaccurate queries returned without error but did not match the MySQL baseline. Error queries include queries reported as not supported as queries that returned an explicit error.

of time preparing for the pilot. Furthermore, MIT LL believes that all instances of inaccurate results are the software implementation not parsing the query or not knowing how to answer the query. MIT LL believes no inaccurate results were returned due to a failure of the underlying technology. At this time there is no standard query language for SPAR implementations. The creation of such a language is a recommended extension (see Section 1.7).

5.3 Database Modification

This section describes the modification capabilities of SPAR implementations at the time of the pilot demonstration. The pilot demonstration had more extensive modification testing than the research program. Modification capabilities during the research program are omitted.

SPAR implementations support database modification. No current implementation handles multiple tables so this requirement is limited to a single database table. All implementations support three basic modifications: UPDATE, DELETE and INSERT.

DELETE Removes a record(s) from the database. Current SPAR implementations support deletion of a single record at a time identified by a primary key.

UPDATE Changes value(s) of some field(s) for a set of records. Current SPAR implementations support updates of a single record at a time identified by a primary key. Some SPAR technologies support updates by marking the original record as deleted and inserting a new updated record.

INSERT Creates a new record in the database table. Current SPAR implementations support inserts as long as the table has a primary key.

The three SPAR technologies have different approaches to handling inserted records. Standard database applications often have two separate mechanisms to handle new records: 1) single record and 2) batched. In either of these modes the database should move from the pre insert state to the post insert state. It should never be possible to query the database in an intermediate state.

SPAR Technology	Data Structure	Number of Records	Time of Processing
BLIND SEER	Preallocated Bloom filter tree	Single	Immediate
ESPADA	Unpopulated forward and reverse indices	Single	Immediate
Stealth	Two unpopulated B-trees of different size	Multiple	At set intervals

Table 11: Summary of approaches for handling inserted records.

Current SPAR implementations do not support both of these modes. BLIND SEER and ESPADA insert a single record at a time. Stealth inserts a batch of inserted records after a configurable period of time. All SPAR implementations current place inserted records in a separate "insert" data structure that is searched separately. These approaches are described in Table 11.

5.4 Policy Enforcement

In this section, we describe the types of policies that each SPAR technology supported by the end of the pilot program. For a simple query like `field = value`, we describe restrictions that each policy can place on the field and value being searched. We reiterate our belief that any policy enforcement mechanism *could* be modified to operate with any other technology's query processing ability. We conclude this section with the results of policy enforcement during the pilot.

BLIND SEER BLIND SEER supports a rich set of query check policies. Policies can apply directly to individual terms in a query or to the combination of terms in a Boolean query. We detail the specific supported policies for each query form below.

For a single-field equality or disjunction search, the policy can restrict specific field names or keyword values based upon a whitelist or blacklist. This policy can be applied separately for each term in a disjunction.

For a conjunction, the policy can enforce a whitelist or blacklist applied to each term individually. More interestingly, the policy can enforce several constraints between terms: (1) if field A is present, then the query must also contain field B, (2) a query cannot search for both keyword X and keyword Y, (3) if keyword X is contained in a query, then field A must also be searched.

For more complicated Boolean queries, such as a CNF or DNF query, any conjunctive or disjunctive policy can be applied separately to each clause. Additionally, for a range query, the policy can restrict the endpoints of the queried range to be wholly contained within a certain interval.

ESPADA ESPADA's policy enforcement mechanism is simpler than BLIND SEER's: it can restrict the fields being searched in a query but not the values being searched. Additionally, the policy can restrict the *order* of clauses in a Boolean expression, which can influence the performance and security of queries.

ESPADA's policy mechanism has special features for two types of queries. First, it can limit the maximum size of any range query; that is, it can specify the maximum difference between endpoints without imposing any restrictions on the actual values of the endpoints. Additionally, the policy can impose a minimum bound on the length of the string in any subsequence or wildcard query.

Optionally, ESPADA supports a "warrant-based" authorization where a separate *judge*, and not the data owner, sets the policy. Independently, ESPADA also supports a "voucher" model where a separate entity can issue vouchers to whitelist or blacklist specific *values* (not just fields). This can occur whether the authorization check is performed by the data owner or the judge described above.

Stealth Stealth's query authorization policy mechanism has the simplest capability of the three. It can enforce blacklists (but not whitelists!) over the query type, the field being searched, and the value being searched in a query. For a range query, the blacklist is over the specific set of endpoints being searched: that is, blacklisting the range [2,6] does not preclude someone from searching for [1,3], [3,5], or [1,7].

Policy enforcement during the pilot We now turn to the ability of SPAR technologies to enforce the pilot policy described in Section 3.2.6. Based on the the reported capabilities of the three implementations, BLIND SEER was the only system capable of enforcing query-check rules for field inclusion and the value searched for.

We issued 909 queries on each SPAR implementation in the pilot risk reduction environment our synthetic data. All queries were either simple conjunctions or in DNF form. Of the 909 queries, 164 were valid, meaning that they should have been accepted. We divide each valid query into two categories – simple conjunctions or DNF. Of our valid queries, 63 were conjunctions and the other 101 were in DNF. The remaining 745 queries were invalid and were divided into four categories: 1) Boolean Field Omitted 2) Boolean Field with Wrong Value Specified 3) Range with Wrong Value 4) Too Large a Range.

Detailed results for each SPAR implementation are in Section F. MIT LL's assessment of each SPAR implementation's current ability to the pilot business rules are in Table 12. SPAR implementations did not exactly match the believed technology capabilities described above. BLIND SEER incorrectly rejected all valid queries in DNF. ESPADA also rejected many valid DNF queries. Furthermore, ESPADA struggled in rejecting invalid queries because it could only do enforcement on query structure and not on values. Stealth performed the best at accepting valid DNF queries, but could not check for the inclusion of a field and thus struggled to reject invalid queries. BLIND SEER did well in rejecting invalid queries, but this may just have been a continued manifestation of the inability to accept DNF queries. No system exactly enforced the desired query-check policy. Also, each system had a different

Ability to support Tested Business Rules	**Blind Seer**	**ESPADA**	**Stealth**
Rule 0 (Allowing Valid Queries)	Low	Low	High
Rule 1 (Inclusion of Boolean Value Set to 0)	High	Low	Low
Rule 2 (Inclusion of Date Set to Proper Range)	High	Low	None
Rule 3 (Restriction of Date to Small Range)	High	High	None

Table 12: Current ability to enforce Business Rules. Based on Tables 26, 33, and 40.

languages for defining their policy, with ESPADA being the most involved at 600 lines to represent the query-check policy used in this pilot demonstration.

6 Performance

As described in Section 3, the SPAR research program and pilot demonstration had different objectives and methodologies. The SPAR research program focused on repeatable and precise testing that was "white-box." Different tests were designed to explore the important variables for each SPAR technology. This precludes direct comparison of results.

In contrast, the SPAR pilot demonstration was designed to measure suitability of SPAR technologies' to a specific use case. When designing the pilot demonstration, MIT LL did not consider the features or limitations of SPAR technologies. All queries were issued to each SPAR technology, even if the SPAR technology was known not to handle this type of query. Results from the research program should be used when projecting SPAR technologies to a new use case. Results from the pilot demonstration should be used when considering SPAR technologies in the demonstrated use case. Since the same methodology was followed for all SPAR technologies in the pilot demonstration, these results can and are compared throughout this report.

6.1 Research Program Performance

During the SPAR research program, MIT LL tailored unique tests for each SPAR technology. In particular, MIT LL tailored the queries and policies tested to the (1) unique capabilities and limitations of each technology and (2) distribution of queries that would showcase the relevant performance characteristics of each technology. As such, the absolute result data (e.g., average query latency) for the different prototype implementations *cannot* be directly compared to each other.

Instead, MIT LL installed a non-privacy-protecting database system using Transport Layer Security (TLS) as a baseline for comparison. Specifically, we executed all test queries on MySQL 5.5 in Phase 1 and on MariaDB 5.5.32 in Phase 2.

To test performance at scale, SPAR implementations were empirically evaluated on database sizes up to 10 terabytes with 100 million records (see Table 2 and Appendix C for details). Their performance figures were captured on MIT LL's test environment that was custom-designed for the testing.

We purposely do not convey the full breadth of MIT LL's testing in this section. We refer interested readers to Lincoln's end of phase reports to the government for complete information. Instead, we choose here to highlight the major strengths and weaknesses of each technology, with only a small subset of our performance results provided as supporting evidence.

The strengths and weaknesses of each technology are intricately related to their design, as described in Section 2 and reviewed below. Fundamentally, each SPAR technology is designed to support a single 'base' query type (either conjunctions or range queries). Then, the technology simply retrofits all other queries into the base type. This section examines the consequences of each team's design decisions upon the performance of their prototype implementation.

6.1.1 BLIND SEER

This section describes BLIND SEER's SPAR prototype as of the end of Phase 2 of the SPAR research program, based upon MIT LL's report to the government at the end of the phase [56].

Base query type The base query type for BLIND SEER is a conjunction of simple keyword searches. Recall that BLIND SEER's index mechanism is structured as a tree of Bloom filters. The filters at the leaves contain the searchable data present in a single database record, and internal leaves represent the union of all data in its children. The searching mechanism proceeds as follows: for each Bloom filter traversed (starting with the root), the searching mechanism checks if each of the desired keywords is present in the Bloom filter; searching proceeds to the node's children if and only if this is true.

Intuitively, this search checks different subsets of the database, starting with the entire set at the root. If all of the keywords in a conjunction are present *somewhere* in this subset, the search proceeds to check whether the terms are all present in smaller subsets. The performance of this indexing mechanism relies crucially on the rarity of 'bad subsets' whose records *collectively* contain all of the keywords but for which no *individual* record matches the query.

For concreteness, let's consider the best- and worst-case scenarios for a single query '`fname = John and lname = Smith`.' The fastest scenario is one in which there is a single John Smith, with no other Johns (who are not Smiths) or Smiths (who are not Johns). In that case, the searching mechanism traverses through the tree in a straight line from the root to the single leaf node corresponding to John Smith. The worst-case scenario is a database in which every odd-numbered entry is a John who is not a Smith, and every even-numbered entry is a Smith who is not a John. In this case, no records actually match the query. However, the query traverses the *entire* tree to learn this information because all Bloom filters in the penultimate level contain one John and one Smith, so they recommend that the search continue onward. Note that if we merely re-order the database such that the first half contains the Johns and the second half contains the Smiths, then the query terminates quickly at the second level of the tree.

Query performance Based upon BLIND SEER's index mechanism, we expect its query performance to be related to the rate of pruning bad subsets. As a result, we expect the best (and most predictable) performance on single-field equality and disjunction searches that do not require any pruning. For conjunction queries, we expect performance roughly to be related to the number of records in the most-selective clause of the query.

Our performance results match this intuition. Figure 9 describes BLIND SEER's performance overall on single-field equality queries where the tree-based search only walks down paths that lead to desired records. The data shows a clean linear relationship with the main fixed cost being due to the tree traversal and the main variable cost per-record being due to the record retrieval process. MIT LL also partitioned this data by the datatype being searched; we found that (1) the fixed cost per query is slightly higher for strings (0.7 seconds)

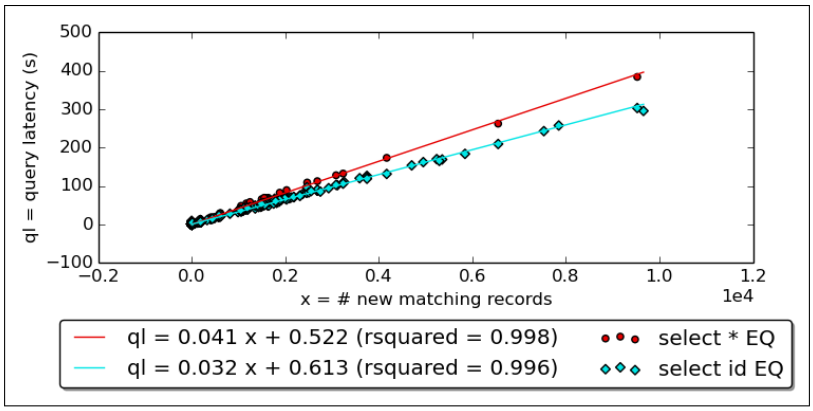


Figure 9: BLIND SEER's performance on single-field equality queries on a 10 TB database as a function of the number of results returned, separated by data type of the field being searched.

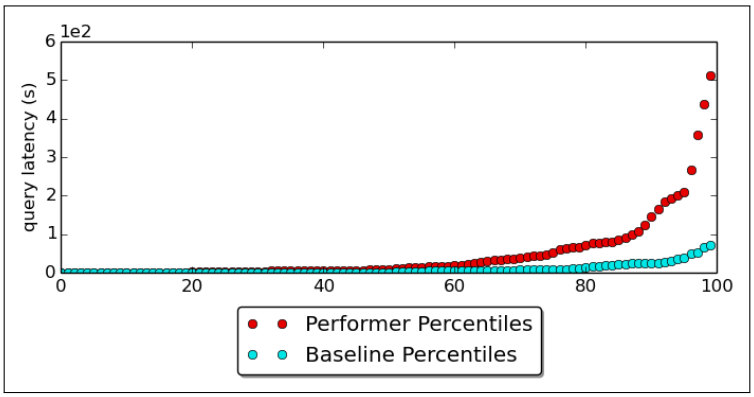


Figure 10: Cumulative distribution function of BLIND SEER's performance on boolean queries executed on a 10 TB database during Phase 2 testing. Note the 100× multiplier for labels on the y-axis.

than it is for integers and dates (0.4 seconds) and (2) the variable cost per-record returned is identical for all datatypes.

Figure 10 provides some insight into the complicated nature of BLIND SEER's performance on `select id` boolean queries. Essentially, their tree-based search mechanism performed relatively well on simpler queries. But, when we fed more complicated queries with several clauses that each match a large fraction of the database, BLIND SEER's tree-based search method fared poorly because it is unable to prune the tree quickly. In total, we observed that we observed that the the number of records in the most-selective clause of the query describes approximately 62% of query performance. The remaining variation is due to the nuanced tree traversal process, whose pruning rate depends on both the distribution (i.e., how many Johns are also Smiths) and location (i.e., whether the Johns are next to the Smiths) of data in the database.

10 TB database		100 GB database	
`select *`	`select id`	`select *`	`select id`
Good: 100% passing		Good: 100% passing	
EQ	EQ		
P3	P3	P3	
P4	P4	P4	
P8	P8	P8	
P9	P9		
		P11	P11
Fair: $>$ 50% passing		Fair: $>$ 50% passing	
		EQ(83%)	
	P1 (86%)		
Poor: $\leq$ 50% passing		Poor: $\leq$ 50% passing	
			EQ (45%)
P1 (8%)		P1 (11%)	P1 (6%)
P2 (33%)	P2 (0%)	P2 (16%)	P2 (1%)
			P3 (7%)
			P4 (11%)
			P8 (8%)
		P9 (50%)	P9 (14%)

Table 13: Percent of each of BLIND SEER's query types that meet the BAA's Phase 2 performance requirement of $5 \cdot \text{baseline} + 8$ seconds, for each database size (either 10^8 or 10^6 records, at 10^5 bytes per row) and `select` clause tested during Phase 2. Note that P11 type queries were only tested on the 100GB database.

Finally, we examine performance for other query types. Table 13 shows the percentage of BLIND SEER's tests that met the BAA's performance requirement of 5 times the baseline plus 8 seconds.[15] BLIND SEER also fared relatively well on keyword (P3) and stemming (P4) query types that are implemented similarly to equality queries. BLIND SEER performed poorly for range (P2) queries due to their ad-hoc method of answering range queries as a complicated conjunction query whose clauses each individually match a large number of records (a method that the baseline need not employ). Boolean queries containing a negation perform similarly poorly because negations are turned into range queries.

Threshold (P8), ranking (P9), and XML (P11) queries are not natively supported by our baseline software and were implemented with a (slower) customized function. Thus, we recommend caution when judging BLIND SEER's performance on P8, P9, and P11 queries.

Finally, we highlight the scalability of BLIND SEER. Based on Table 13, BLIND SEER generally performed better in comparison to the baseline on the 10 TB database than it did on the 100 GB database. Additionally, BLIND SEER performed better on `select *` queries

[15]Query types are reviewed in Section 5.

than it did on `select id` queries. In summary, the overhead of BLIND SEER's index search mechanism is less damaging (relative to the baseline) on more 'complicated' queries on which the baseline must work harder too.

We note that, due to software installation and runtime issues by the BLIND SEER prototype, MIT LL was only able to execute query tests on the 10TB database (10^8 rows, 10^5 bytes per row) and 100GB database (10^6 rows, 10^5 bytes per row). Both `select *` and `select ID` queries were run against both database sizes.

Policy performance As a reminder, BLIND SEER supports a rich set of policies to authorize queries. Policies can apply directly to an individual keyword search or to the combination of keyword searches included in a Boolean query (see Section 5.4 for details). In both phases, MIT LL tested BLIND SEER's correctness and performance on policy tests executed on a 10 GB database with 10^5 rows and 10^5 bytes per row.

BLIND SEER was successful in enforcing the a variety of policies that exercise the different features available in their robust policy language. For all finished tests, BLIND SEER correctly rejected queries according to the specified policy. Due to time limitations, the full breadth of policies across DNF/CNF query types was not tested. Additionally, the policy checker also exhibited undiagnosed stability issues on a specific conjunctive policy test.

While the latency overhead associated with policy enforcement was not formally tested, MIT LL noted that queries that failed the specified policy took noticeably longer than the same queries run without a policy in place.

6.1.2 ESPADA

This section describes ESPADA's SPAR prototype as of the end of Phase 2 of the SPAR research program, based upon MIT LL's report to the government at the end of the phase [57].

Base query type ESPADA's search mechanism is optimized to handle conjunction queries in a special form that the designers call 'ESPADA normal form,' which has the format $w \wedge \phi(x_1, x_2, \ldots, x_n)$ where the 'start term' w and all 'cross terms' x_i are single-field equality searches and ϕ is an arbitrary boolean formula over the cross terms (for instance, it can negate any of the cross terms). ESPADA's inverted index search mechanism first determines the set of records that match the start term w. Then it searches *only these records* (using a second data structure) to determine whether or not they match the cross terms x_i.

Query performance Due to their index structure, we expect ESPADA's performance on conjunction queries to correlate strongly with the number of records that match the start term. MIT LL's testing results confirm that failing to adhere to this rule causes grave damage. For the query `ssn = 123456789 and sex = female`, we found that swapping the order of the clauses increased query time from less than a second to a full day. As a result, MIT LL's automated query generator ensured that the most selective clause was placed first

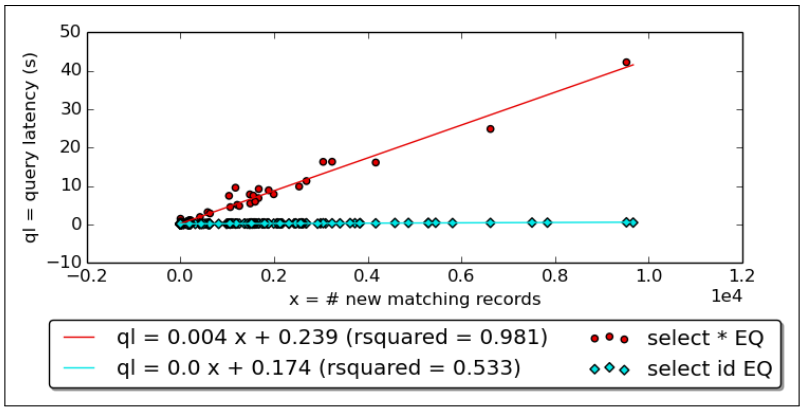


Figure 11: ESPADA's performance on single-field equality queries on a 10 TB database as a function of the number of results returned, separated by data type of the field being searched. Note the $10^4\times$ multiplier on the x-axis.

in a conjunction query. Hence, all results that follow should be viewed under the (non-trivial!) assumption that a database querier is aware of the data distribution and capable of making intelligent decisions about the order of clauses in a query.[16]

Our test results show that ESPADA's index retrieval mechanism is incredibly fast, even performing faster than the baseline on occasion. Query processing in ESPADA is multi-threaded with asynchronous I/O. ESPADA's record retrieval and transmission procedure relies upon the use of a customized filesystem layout to maximize disk seeks, which can be incredibly fast if tuned properly (but see our caveat below). With a fast procedure to return unstructured data, ESPADA tends to perform faster than the baseline on our wider 10^5 byte per record schema (with a large fingerprint blob) but slower than the baseline on the narrower 10^2 byte per record schema.

Obtaining performance data for ESPADA on `select id` queries is challenging due to the sheer variety of variables that govern the performance of their inverted-index lookup mechanism; at the speeds they are able to achieve, even the location of the disk head at the start of the query is a pertinent variable. Figure 11 demonstrates that `select *` equality queries on the 10 TB database are much more predictable.

Table 14 summarizes of the best-fit curves for the query latency as a function of number of records returned for each database size and query type. We also provide coefficients of determination: they show that ESPADA's query performance is more reliable for single-field equality (including keyword and stemming) and boolean query types than it is for more complicated query types (range, wildcard, subsequence) that get transformed internally into a complicated boolean query.

We caution that ESPADA's database initialization is both slow and rather complicated due to their customized filesystem layout. During phase 1 testing, we needed to reformat our

[16]During the pilot demonstration, no effort was made to transform queries into ESPADA normal form. There were a small number of queries that were not easily expressible in ESPADA normal form. These queries had by far the worst performance. See Section 6.2 and Figures 13 and 14.

# of records	`select`	Query type	Best-fit function	r^2
10^5	*	equality	$ql = 0.002 \cdot x + 0.16$	0.807
10^5	*	boolean (P1)	$ql = 0.001 \cdot x + 0.289$	0.972
10^5	*	range (P2)	$ql = 0.001 \cdot x + 0.275$	0.925
10^5	*	keyword (P3)	$ql = 0.001 \cdot x + 0.176$	0.974
10^5	*	stemming (P4)	$ql = 0.001 \cdot x + 0.152$	0.908
10^5	*	wildcard (P6)	$ql = 0.001 \cdot x + 0.262$	0.125
10^5	*	subsequence (P7)	$ql = 0.001 \cdot x + 0.288$	0.522
10^6	*	equality	$ql = 0.001 \cdot x + 0.226$	0.989
10^6	*	boolean (P1)	$ql = 0.001 \cdot x + 0.274$	0.921
10^6	*	range (P2)	$ql = 0.001 \cdot x + 0.258$	0.989
10^6	*	keyword (P3)	$ql = 0.001 \cdot x + 0.216$	0.998
10^6	*	stemming (P4)	$ql = 0.001 \cdot x + 0.198$	0.999
10^6	*	wildcard (P6)	$ql = 0.002 \cdot x + 0.479$	0.646
10^6	*	subsequence (P7)	$ql = 0.004 \cdot x + 1.149$	0.268
10^8	*	equality	$ql = 0.004 \cdot x + 0.239$	0.981
10^8	*	boolean (P1)	$ql = 0.004 \cdot x + 0.418$	0.973
10^8	*	range (P2)	$ql = 0.004 \cdot x + 0.355$	0.99
10^8	*	keyword (P3)	$ql = 0.004 \cdot x - 0.023$	0.967
10^8	*	stemming (P4)	$ql = 0.004 \cdot x + 0.407$	0.985
10^8	`id`	equality	$ql = 0.0 \cdot x + 0.174$	0.533
10^8	`id`	boolean (P1)	$ql = 0.0 \cdot x + 0.328$	0.032
10^8	`id`	range (P2)	$ql = 0.0 \cdot x + 0.343$	0.662
10^8	`id`	keyword (P3)	$ql = 0.0 \cdot x + 0.204$	0.848
10^8	`id`	stemming (P4)	$ql = 0.0 \cdot x + 0.185$	0.929
10^8	`id`	wildcard (P6)	$ql = 0.012 \cdot x + 34.117$	0.04
10^8	`id`	subsequence (P7)	$ql = 0.008 \cdot x + 31.623$	0.01

Table 14: Best-fit curves for ESPADA's query latency (ql), as a function of the number of records in a database (x), the `select` clause type, and the query type. Results are shown for the 10^5 bytes per record schema only. Statistical coefficients of determination (r^2) are also provided.

RAID array to change its file block allocation. We note that this is an extremely low-level change that is not (to our knowledge) documented or automated anywhere. Hence, the performance numbers during the SPAR research program (as good as they are) must come with a corresponding concern about the usability and software maturity of the technology. The pilot effort provides a valuable benchmark here of MIT LL performing software installation without input from the ESPADA team; see Table 16 and Appendix F.2 for details.

Policy performance ESPADA supports simple query authorization policies that restrict the fields that may be searched, plus some extra features on range and subsequence queries. In both phases, MIT LL tested BLIND SEER's correctness and performance on policy tests executed on a 10 GB database with 10^5 rows and 10^5 bytes per row. ESPADA was successful in enforcing all policies that we tested them on. Policy enforcement added a small constant latency that did not vary with database size.

6.1.3 Stealth

This section describes Stealth's SPAR prototype as of the end of Phase 1 of the SPAR research program, based upon MIT LL's report to the government at the end of the phase [54]. Because Stealth only participated in Phase 1, whereas the two teams above participated in both phases of the research program, we strongly caution readers not to compare the teams directly based upon the data presented here. The Stealth team has since made substantial changes to their technology and corresponding implementation. Stealth added new query types during the pilot effort, as described in Section 5.1.

Base query type The Stealth prototype reduces all supported query types to range queries. Recall that Stealth's technology focuses on securing a B-tree index structure of pointers to all records in sorted order. As a result, range queries can be performed using two B-tree lookups to find pointers to the records at the start and end points of a range; then, all records between these two pointers are returned to the querier. Single-field equality searches can be performed similarly with only one B-tree lookup.

The other query types that Stealth handled in Phase 1 (keywords, stemming, and subsequence queries) were all computed by converting them into a range or single-field equality query as appropriate; see Section 2.3 and Stealth's proposed query type document [43] for more details. During the research program, Stealth lacked the ability to perform conjunction queries or anything that would further refine the set of records to return within a range; this restriction was addressed during the pilot effort.

Stealth provided an informal description of Boolean queries [38]. They maintain the basic structure of a separate B-tree per field. A set of record pointers is retrieved from each B-tree. A private set intersection protocol is performed on the record pointers for a conjunction, and a private set union is performed for a disjunction. Negations are not currently supported. This approach leaks the size of how many records match each individual query term.

Select type	Query type	Database size	Best-fit line, in seconds	Coefficient of determination
`select *`	all	10^4 rows (1 GB)	$3.731 + 0.09403x$	$r^2 = 0.992$
`select *`	all	10^6 rows (100 GB)	$5.997 + 0.08303x$	$r^2 = 0.995$
`select *`	all	$5 \cdot 10^6$ rows (500 GB)	$6.116 + 0.08852x$	$r^2 = 0.999$
`select *`	all	10^8 rows (10 TB)	$13.13 + 0.08216x$	$r^2 = 0.870$
`select id`	all	10^4 rows (1 GB)	$2.283 + 0.07663x$	$r^2 = 0.998$
`select id`	all	10^6 rows (100 GB)	$5.631 + 0.07676x$	$r^2 = 0.989$
`select id`	all	$5 \cdot 10^6$ rows (500 GB)	$5.655 + 0.07657x$	$r^2 = 0.999$
`select id`	all	10^8 rows (10 TB)	$11.12 + 0.07731x$	$r^2 = 0.969$
`select *`	P2	10^8 rows (10 TB)	$19.23 + 0.08455x$	$r^2 = 0.267$
`select *`	P3	10^8 rows (10 TB)	$10.35 + 0.09169x$	$r^2 = 0.987$
`select *`	P4	10^8 rows (10 TB)	$10.36 + 0.08103x$	$r^2 = 0.912$
`select *`	P7	10^8 rows (10 TB)	$22.31 + 0.08083x$	$r^2 = 0.942$

Table 15: Stealth's phase 1 query latency results as a function of the number of results to each query ('x'). The first two sets of results are shown as a function of database size, averaged over all query types. The final set of results is displayed instead as a function of query type for only the largest database tested.

Query performance The nature of Stealth's B-tree search made its performance more straightforward than the baseline or any other SPAR technology. Query latencies essentially comprised two factors:

1. One or two scans through the B-tree (taking approximately 13 seconds on our testbed). The cost of a scan depends primarily on the size of the B-tree, which itself depends on the database size. Whether 1 or 2 scans are required depends on the query type: keyword (P3) and stemming (P4) queries require one scan whereas range (P2) and subsequence (P7) queries require two.

2. A fixed cost per record retrieved (approximately 0.08 seconds, on our testbed).

The results of Phase 1 tests are summarized in Table 15. We emphasize the fact that query latency times are highly predictable based solely on knowledge of the database size, without knowing anything else about the query type or values being searched. This property is decidedly *not* true of the other SPAR technologies nor of our baseline databases, and it gets to the heart of the difference between Stealth and other database technologies.

First, most databases perform extensive pre-computation to build indices that can be searched as quickly as possible, especially for common results such as single-field equality matches. In contrast, Stealth's focus was on designing indices that can be searched as securely as possible, and their low leakage is balanced by their long search types even when a single value is returned. Second, most databases enable conjunction queries that permit the server to prune results as much as possible before transmitting them back to the querier.

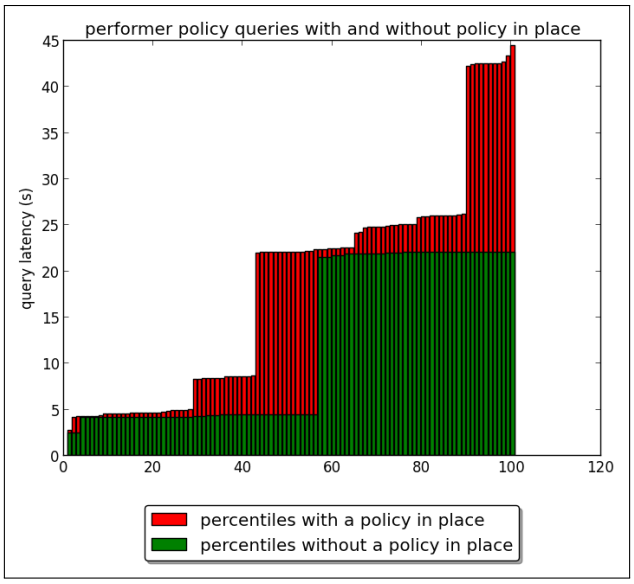


Figure 12: Cumulative distribution function of Stealth's query latency during our phase 1 policy testing, with and without policies enabled.

In contrast, Stealth's technology was incapable of performing a 'secondary' search inside the results of a range query, and instead must perform the slower and more predictable approach of returning all records that match a range query.

As a result, Stealth performs poorly on the 'simple' types of queries that people wish to execute in practice, such as single-field equality searches, which modern databases can handle incredibly quickly (at the millisecond level). Stealth performs much better, relative to the baseline, on more 'complex' queries like subsequence or `select *` queries.

Policy performance During the research program, Stealth's query authorization policy was performed using a generic secure multi-party computation protocol. As a result, its policy checking mechanism was very slow, introducing a noticeable delay in the query protocol up to 25 seconds as shown in Figure 12.

6.2 Pilot Demonstration Performance

This section describes the performance of SPAR technologies during the pilot demonstration. The three SPAR technologies were subject to the same set of tests so we directly compare performance in this subsection. The three main tests were 1) query latency times 2) insert times and 3) deletes times. Query latency times were remeasured after records were added to the database. Throughout this subsection, the baseline for comparison is MySQL 5.1 with single column indices for all searchable columns.

Query-Response Speed Queries were categorized as either *simple* and *complex*. Complex queries either had a mix of "AND" and "OR" expressions or contained a range. Query timing was measured across this categorization, with each of the 202 queries in the query corpus being categorized. SPAR implementations occassionally responded more quickly than the MySQL baseline, while in other cases it response times were several orders of magnitude slower. Figure 13 depicts timing for simple queries, and Figure 14 depicts timing for complex queries.

ESPADA demonstrated very fast query-response times on specific query formats, and not just on simple queries. ESPADA performed particularly well on queries in ESPADA normal form, in which the first term of each clause was very selective. However, if the query could not be converted to ESPADA normal form, ESPADA was very slow. As shown in Figures 13 and 14, Stealth's implementation was the most consistent of the three across simple and complex queries. It was also often the fastest. Finally, BLIND SEER demonstrated the slowest query-response times, with a high variability between complex and simple queries. This is depicted in Figure 14 and Table 16. The BLIND SEER development team has indicated that after the pilot demonstration, they have improved their speed performance at least five-fold and that they can effect further improvements [21]. MIT LL has not evaluated this claim. Stealth's median and mean query response times are almost the same. The means of both BLIND SEER and ESPADA are much larger than their respective median query response times. This indicates that there were a few queries for both BLIND SEER and ESPADA with much slower query response times.

For interactive use-cases (including the pilot use case) SPAR technology demonstrated sufficient query response times in comparison to the MySQL baseline. SPAR technology query response times are probably not sufficient for use in statistical use cases. However, SPAR technology was not designed for these types of applications.

Inserts SPAR implementations were tested on their update capabilities, both inserts and deletes. This analysis measured the total time taken for inserts – starting from the time the insert process was initiated and ending when the records appeared in the database. On this metric, Stealth performed very well on insert speed, as shown in Figure 15. Stealth was also the only implementation able to scale up to one million records. ESPADA performed the worst.

The pilot demonstration also tested the impact of inserts on subsequent query-response time. Figures 16 and 17 depict timing for simple and complex queries respectively. The impact on inserts in ESPADA was fairly limited, whereas Stealth was greatly impacted. BLIND SEER's post-insert performance is not representative due because the post insert run of the query corpus failed to complete. Among the failed queries were several of the worst performing preinsert queries.

Insert Testing Caveats Insert speed between the three SPAR technologies is difficult to compare. The three technologies handle insertion very differently. Stealth batched inserts. BLIND SEER pre-allocated space in the data structures for updates and handles inserts

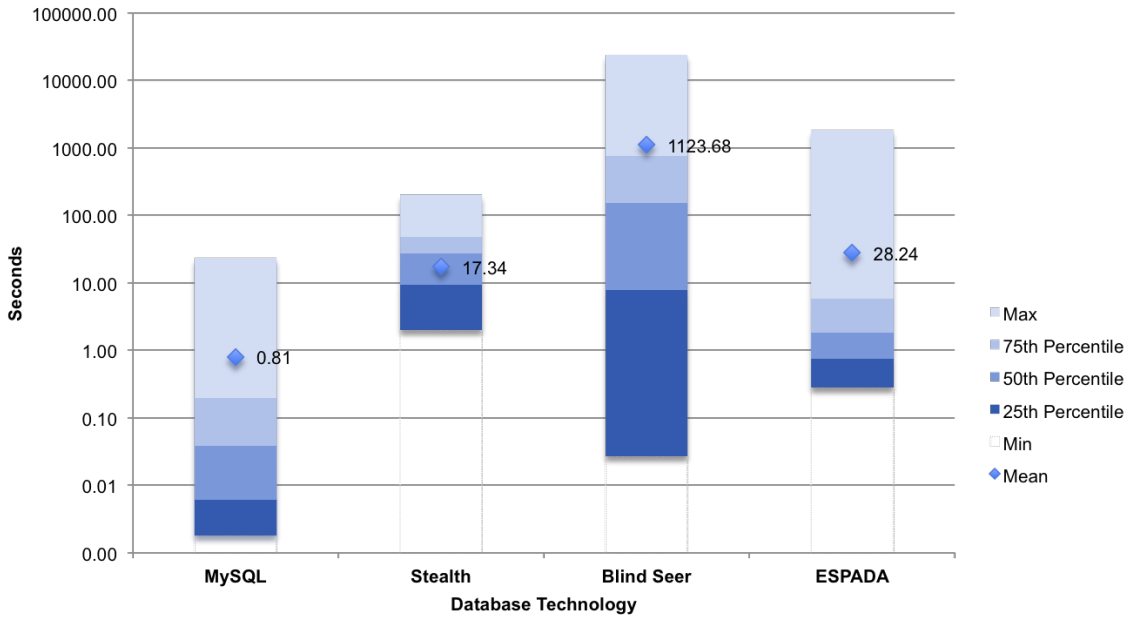


Figure 13: Query-response time for simple queries in seconds (pre-insert)

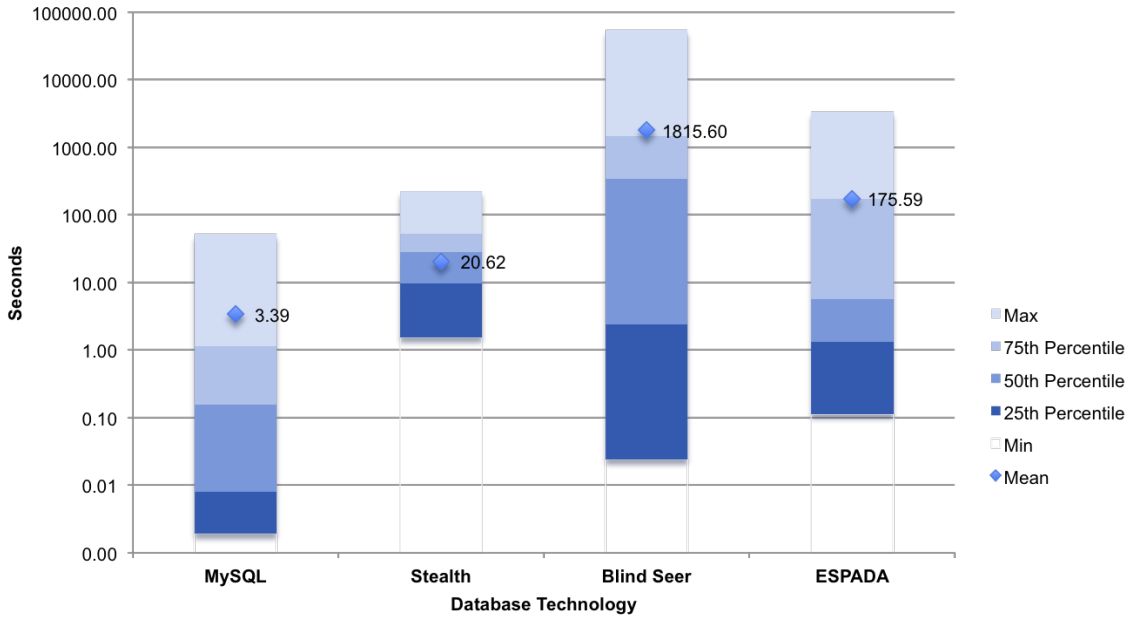


Figure 14: Query-response time for complex queries in seconds (pre-insert)

	Technology	Mean	Min	25th	Median	75th	Max
Pre-Insert Simple Queries	BLIND SEER	1120.0	0.03	7.75	145	608	23200
	ESPADA	28.2	0.28	0.47	1.08	3.96	1840
	Stealth	17.3	2.01	7.32	18.20	20.70	156
	MySQL Baseline	0.81	0.00	0.00	0.03	0.16	23.5
Pre-Insert Complex Queries	BLIND SEER	1820	0.02	2.39	343	1110	53600
	ESPADA	176	0.11	1.21	4.28	166	3220
	Stealth	20.6	1.53	8.24	18.7	24.6	175
	MySQL Baseline	3.39	0.00	0.01	0.15	0.99	51.2

	Technology	Mean	Min	25th	Median	75th	Max
Post-Insert Simple Queries	BLIND SEER	1160	0.03	10.2	132	640	24700
	ESPADA	31.6	0.27	0.49	1.12	3.55	1960
	Stealth	23.8	3.23	12.8	24.6	29.7	153
	MySQL Baseline	.95	0.00	0.01	0.04	0.21	31.0
Post-Insert Complex Queries	BLIND SEER	1080	0.02	2.22	125	1010	9050
	ESPADA	191	0.12	1.24	4.51	176	3430
	Stealth	28.0	2.73	13.7	26.3	34.8	180
	MySQL Baseline	3.42	0.00	0.01	0.15	1.09	51.7

Table 16: Query response times of all three implementations in seconds.

immediately. Inserts in the ESPADA implementation have transactional behavior and are handled immediately. (We further describe the insert approaches in Section 5.) All three systems were tested on inserts of 10 thousand, 100 thousand, and 500 thousand records. The Stealth implementation was also measured on a 1 million record insert. The BLIND SEER system was not measured on 1 million inserts because the system preallocates space for inserted records and the system was only configured with a total capacity of 1 million inserted records. The ESPADA implementation was capable of supporting 1 million inserts but the time to insert 500 thousand records was prohibitive so MIT LL choose not to attempt inserting 1 million records.

Delete Testing Delete-operation timing and accuracy was evaluated in the *risk reduction environment* because its performance was not believed to be data dependent and there was limited test time in the pilot environment. Ten queries were used to select a set of records. Each SPAR implementation was then asked to delete each returned record. Each query was run again and it was confirmed that they now returned no records.[17]

As can be seen in Figure 18, BLIND SEER performed the best on deletions with Stealth not far behind. It is uncertain how both systems would scale in terms of query timing given that BLIND SEER and Stealth are both marking records as deleted but the space used for

[17]However, no attempt was made to ensure that only the specified records were deleted.

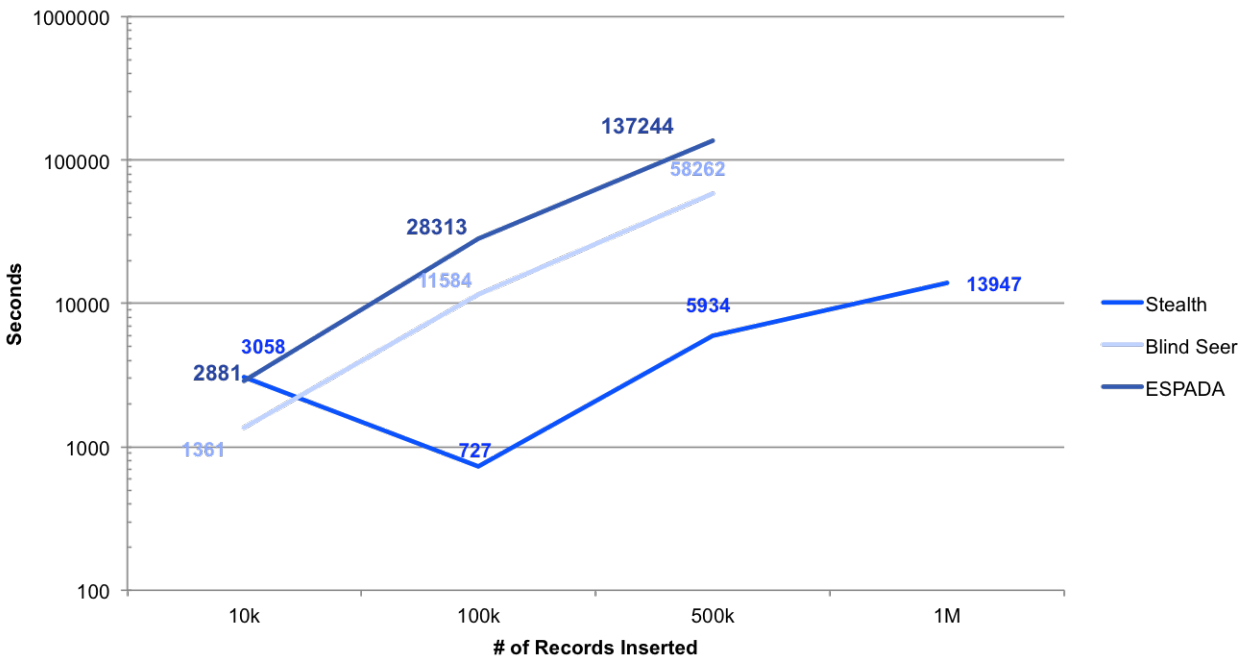


Figure 15: Elapsed time in seconds to number of records inserted

the record is not reclaimed. Of the three, ESPADA is the slowest, again due to the fact that they handle any modification of the database as transactions. Unlike inserted records, in all three implementations deletions take effect immediately.

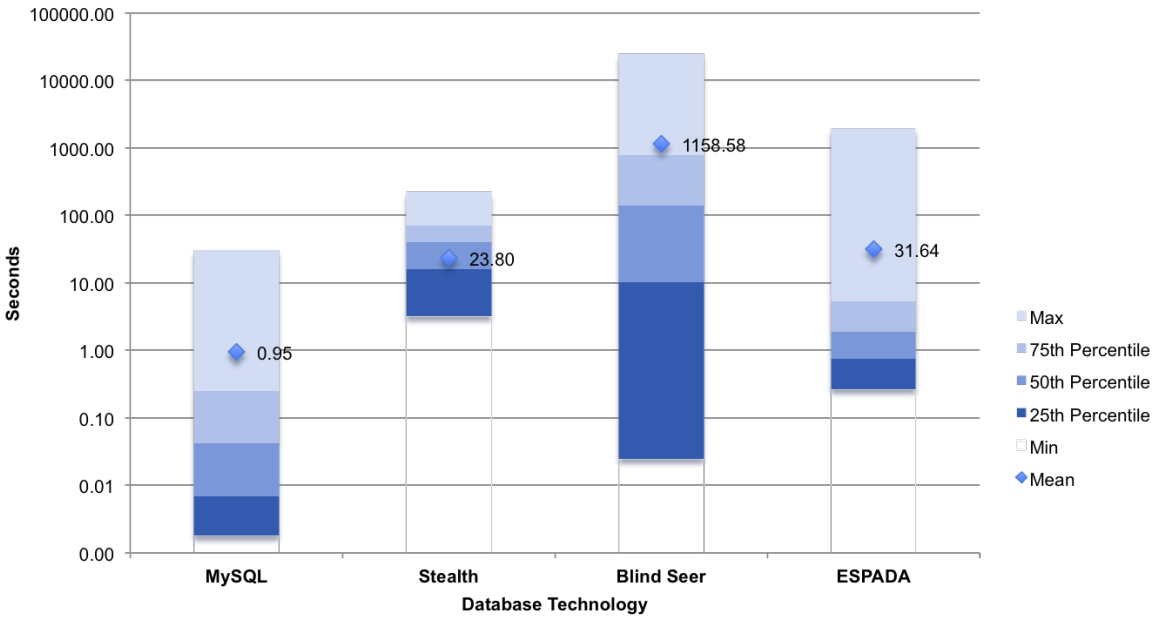


Figure 16: Query-response time for simple queries in seconds (post-insert)

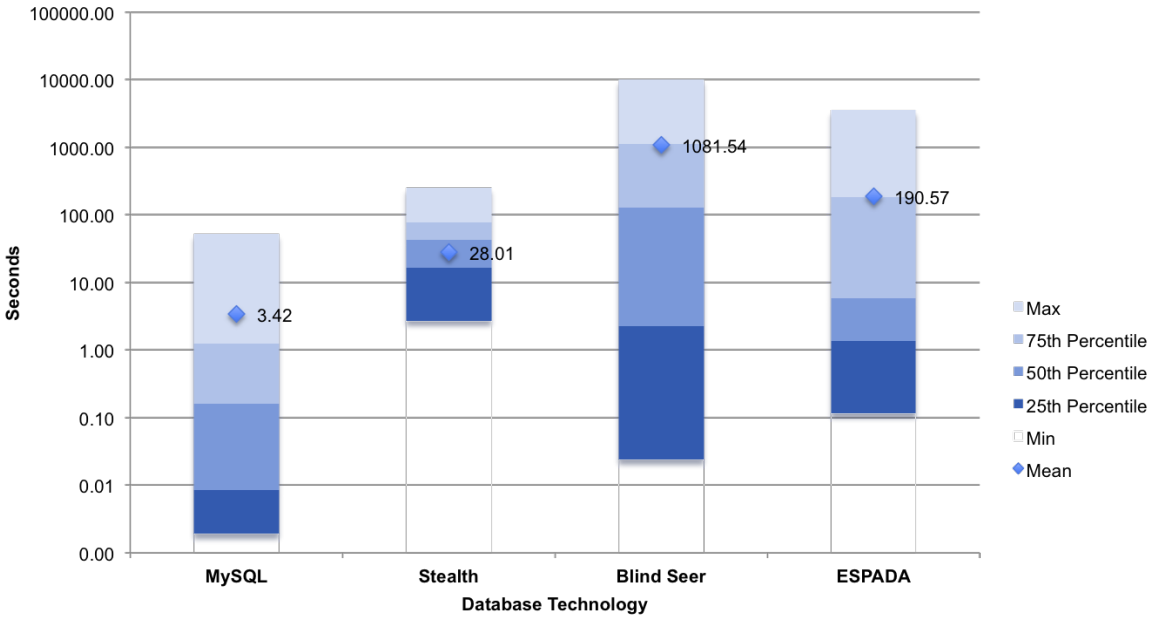


Figure 17: Query-response time for complex queries in seconds (post-insert)

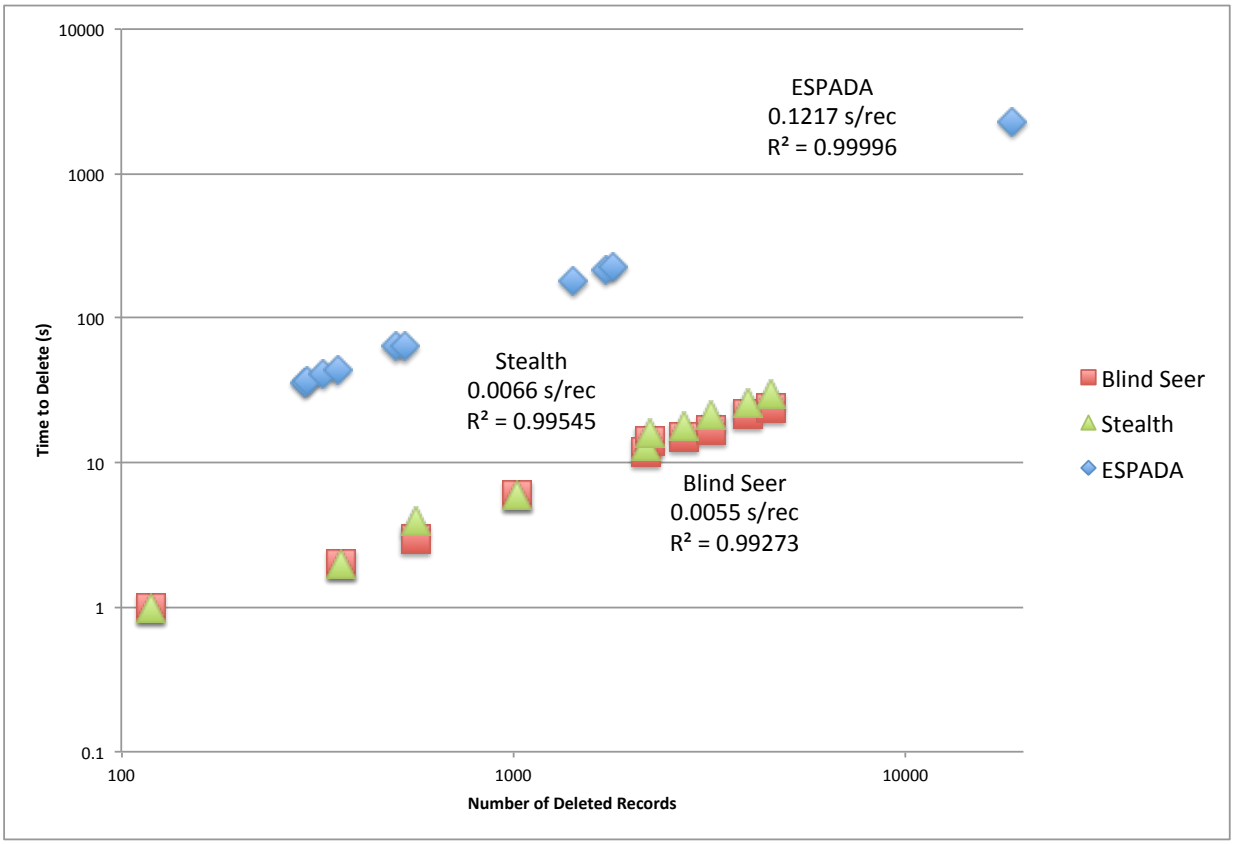


Figure 18: Delete times in seconds

7 Software Maturity

The SPAR program developed cryptographic technology and software implementations designed to demonstrate this technology. The SPAR BAA [22] did not specify requirements for the software implementations. As these implementations move towards deployment, evaluating the software is important. During the SPAR research program, each team was given direct access to MIT LL's test hardware. They were able to customize their software and debug problems with the MIT LL evaluation team in real time. It was difficult to evaluate the maturity of SPAR implementations during the research program.

The pilot demonstration took place in a classified cloud environment. SPAR teams did not have access to this environment. The MIT LL evaluation team deployed, installed, and configured each SPAR implementation. Each SPAR team provided phone and email support as needed.[18] This section contains anecdotal information about that experience.

At their current maturity level, the SPAR implementations are more suited for research than real-world deployment. As far as the authors are aware, the SPAR teams performed all prior deployments of their respective implementations. This section describes the experience of MIT LL in deploying each implementation for the pilot demonstration. The deployment is divided into three sections – installation, configuration, and operation. The comments in this section are based on the experience of the MIT LL team deploying the SPAR implementations in both the risk reduction and pilot environments. The deployment itself used Red Hat Enterprise Linux (RHEL).[19]

7.1 Installation

Installation was fairly straightforward for each system, and Stealth's seemed to be the simplest. All three teams provided custom installation scripts, along with documentation. The installation process utilized the platform's package-management system to download source files for services that would be used by each implementation.

Each SPAR implementation's installation also included source for some external libraries (e.g., for OpenSSL). This approach was adequate for the pilot demonstration. However, in practice, this approach would lead to both difficult maintenance and potential security vulnerabilities. For example, the dependencies provided for source builds of Stealth and ESPADA included cryptography libraries (e.g., gnutls, nettle, and openssl). Consider a hypothetical scenario in which a vulnerability is discovered in one of these libraries and a patch is created by the maintainers of that library. In an enterprise, security patches are typically distributed via the package-management systems of enterprise machines (e.g., Windows Update). A patch distributed in this way would update all system libraries installed

[18] MIT LL test personnel also deployed SPAR implementations in the pilot risk reduction environment. During the risk reduction test SPAR teams had network access to the cloud environment and could provide on computer support. This was not possible in the actual pilot environment.

[19] The risk reduction environment used CentOS 6, there were limited issues moving between the two operating systems. The research program used Ubuntu 12.04. SPAR teams did report issues changing to CentOS 6 due to the older Linux kernel.

via the package-management system. However, it could not patch the in-source builds of these libraries in SPAR implementations.

Previous SPAR testing took place on Ubuntu while this testing used RHEL. This change, along with the aggressive schedule for the pilot demonstration, may have led the developers to take this approach. RHEL has older, more mature versions of most packages because it targets enterprise customers who typically value stability over update frequency. It is quite possible that packages for the required versions of some dependencies were not yet available for RHEL. Bundling and building libraries with their software gave the developers much more control over which specific versions of the libraries were used. Further, the development teams only had a month or two of notice about the specific target platform for the pilot demonstration, and they were using that time to implement other essential features.

BLIND SEER's dependencies were all available as RHEL 6 packages with the exception of a library called Thread Building Blocks, which they included as a source build. None of their dependencies were statically linked. Stealth's dependencies were all provided as source builds and several of these libraries, including security-critical libraries, were statically linked. Consequently, the SPAR software would need to be recompiled each time a patch was applied for one of those library dependencies. The ESPADA team apparently customized the OpenSSL library for ESPADA, which presents even more of a maintenance concern. Any patch that affects the customized code might need to be merged into their customized source build of OpenSSL.

It would be preferable if each system would depend only on standard packages that are available via the package-management system of the target platform. It would also be preferable for libraries to be dynamically linked, particularly those which implement security capabilities. The amount of work to get the correct version of all required dependencies packaged for each implementation on a given target enterprise platform is not known and varies by implementation. BLIND SEER seemed to have the smallest number of dependencies outside the package manager, at least for the target platform.

7.2 Configuration

Overall, Stealth seemed the easiest to configure while the ESPADA and BLIND SEER systems seemed about equal in difficulty. ESPADA had the most complex documentation and the most extensive collection of tunable configuration parameters. While this offered a lot of control over ESPADA's performance, it also contributed substantially to configuration complexity. Stealth performed ingest in the shortest time, while ESPADA took the most time. BLIND SEER's ingest had to be performed multiple times because of parsing problems.

Schema Specification Unlike conventional relational databases, where indexing and tuning can be performed at any time, the SPAR technology required specification of the schema and any indexing requirements before data was ingested. The MIT LL team developed a standard "annotations" file format for SPAR implementations to parse. This file specified field name, field size, data type, minimum value, maximum value, total number of possible

values (if less than the difference between minimum and maximum value), and indexing requirements for each field in the schema. This means a database administrator must anticipate all possible searches ahead of time. Adding an index to a SPAR implementation changes the query times, memory and hard disk footprint. Optimizing index types is an important and delicate process.

While this was sufficient for the pilot demonstration, this approach is limited. Any changes to the schema or the indexing requirements of an operational system would require re-configuring and re-ingesting all data. To achieve comparable modifiability and tuning capabilities with traditional relational databases, more work is required. Of the three systems, Stealth's schema and ingest parsing seemed the most mature and resilient. The other systems' parsing engines halted on simple things such as whether a table name was quoted.

Policy Specification Each implementation team had its own approach to specifying a query-check policy. This was the most error-prone and time-consuming aspect of configuring the systems for deployment. In some cases, the implementation teams had to make code changes to make it possible to express the desired policy. This is untenable as a general solution for specifying a policy to be enforced.

It was fairly easy to specify a query-check policy in BLIND SEER. However, BLIND SEER would only accept queries in very specific formats that needed to be specified. Thus, configuring BLIND SEER to enforce policies effectively and answering all legitimate queries is still difficult. ESPADA had very complex query-check rules that used structural whitelists. The fairly simple policy used in this pilot demonstration became 600 lines in the ESPADA format, with each allowed query structure specifically whitelisted. Stealth had a simplest syntax for specifying query-check policy, though it required a compilation step before starting the system.

7.3 Operation

It is generally considered a best practice for long-running services to be implemented as background processes (e.g., Linux daemons or Windows services). These background processes can typically be managed using the platform's standard service-management tools. Further, they are generally non-interactive; they read configuration information from a standard location and write log information to a standard location. The indexing and policy-checking components of the SPAR implementations, by contrast, were command-line executables that typically wrote to standard output. They had to be run using special utilities to avoid needing to remain logged into the system while the program was running. Such behavior was acceptable to meet the aggressive pilot demonstration schedule, but would not likely be popular with IT professionals tasked with deploying such a system in an operational setting.

8 Usability

Usability was not an explicit requirement of the SPAR BAA [22]. However, usability reflects the current status of SPAR security (Section 4), functionality (Section 5), performance (Section 6), and software maturity (Section 7). MIT LL views usability as a summary of the current state of SPAR technology.

The SPAR research program made no attempt to assess the usability of SPAR technology. Usability is application dependent. The research program instead focused on assessing security, functionality, and performance.

The pilot demonstration included a live human-subjects component. Operators interacted with SPAR technology through a web interface developed by MIT LL. As described in Section 3, operators were not allowed to interact with a baseline technology. Since this was different than the user interfaces that they use in practice, it could have affected their perception of the technology. The lack of a baseline may also hide other impacts of experimental design. Operators who participated in the study were given a questionnaire for each SPAR technology that they used. This questionnaires included Likert questions about agreement with statements about the usability of the SPAR technology. The user tasks and the full questionnaires can be found in the report addendum [10],

Almost all operators took the training session using Stealth technology. Thus, all responses from the training are presented in together in Figure 19. Overall, many operators agreed that the SPAR technology could be incorporated into their workflow. Most agreed that most people could learn to use the technology quickly.

Individual usability results from each implementation are in Figures 20, 21, and 22 for BLIND SEER, ESPADA, and Stealth respectively. These depict agreement with statements made from the second user session and beyond. Operators using BLIND SEER were more likely to disagree with the statement, "I was able to perform my tasks in an acceptable time." Based on discussions with the operators, slow and unpredictable timing is why operators using BLIND SEER were also more likely to disagree with the statement, "I would want to use this technology regularly." This demonstrates that BLIND SEER's query-response time led to usability concerns. Operators were unsure if they should wait for a query to resolve, or if they should do something else while waiting. ESPADA and Stealth were both more favorable to operators – ESPADA often responded more quickly than Stealth, but Stealth was more consistent in how long its queries took.

In summary, the pilot was the first demonstration of SPAR technology in a real use case. The pilot was a first step in transitioning SPAR technology from research-ware to a production capability. The pilot helped identify critical challenges to deploying SPAR technology. MIT LL believes these challnges represent a fraction of the challenges contained in the SPAR vision. We end with the following quotes from operators:

- "The technology was easy to use and would definitely be helpful to the community."
- "Its always useful for analysts to have direct access to data sets that will benefit their work. If the technology is able to streamline the process of obtaining that data, analysts will find applications for it."

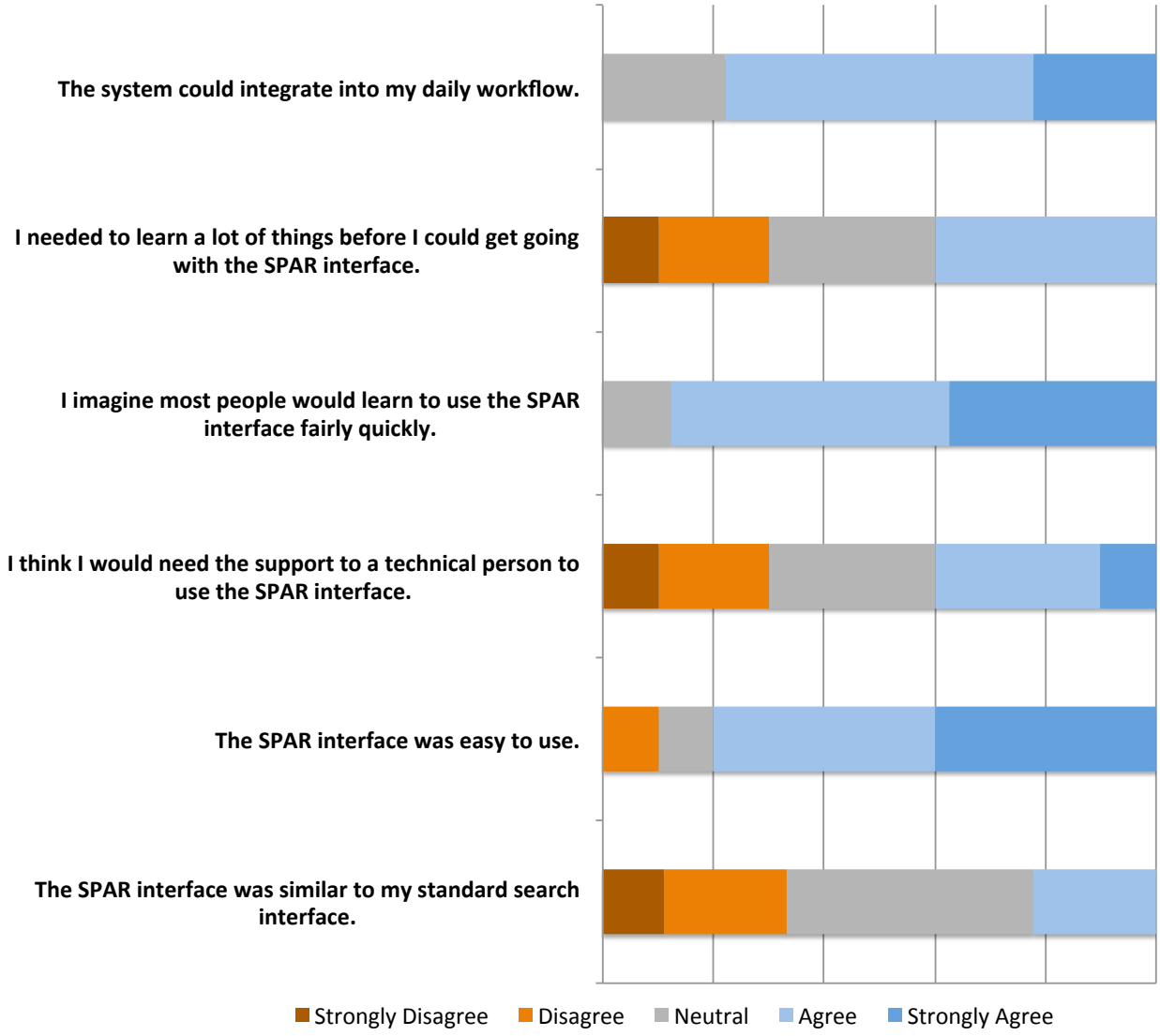


Figure 19: Operator agreement with questions about SPAR technology. These questions were administered during the training session.

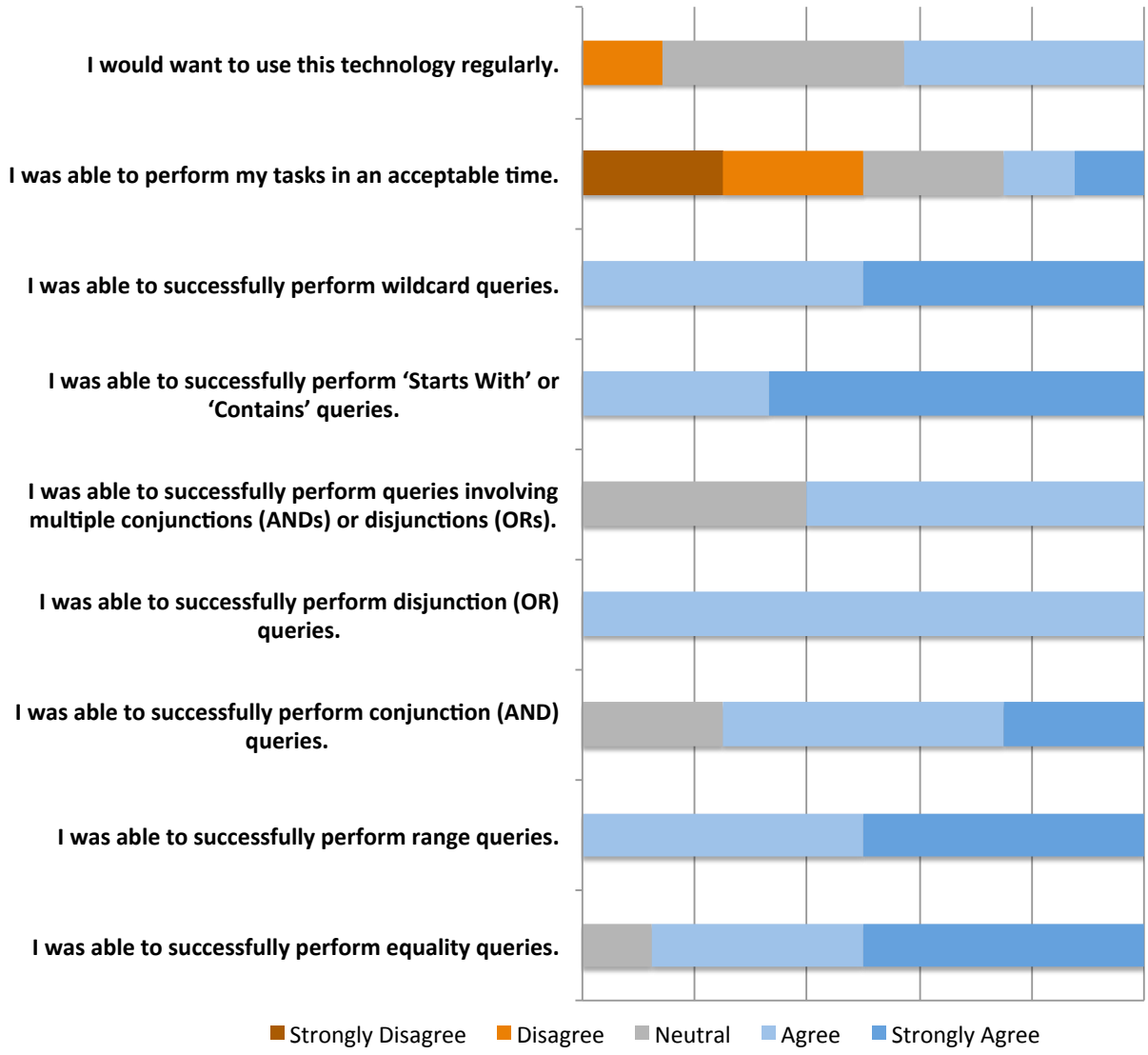


Figure 20: Operator agreement with questions about BLIND SEER. These questions were administered during the scripted user testing session.

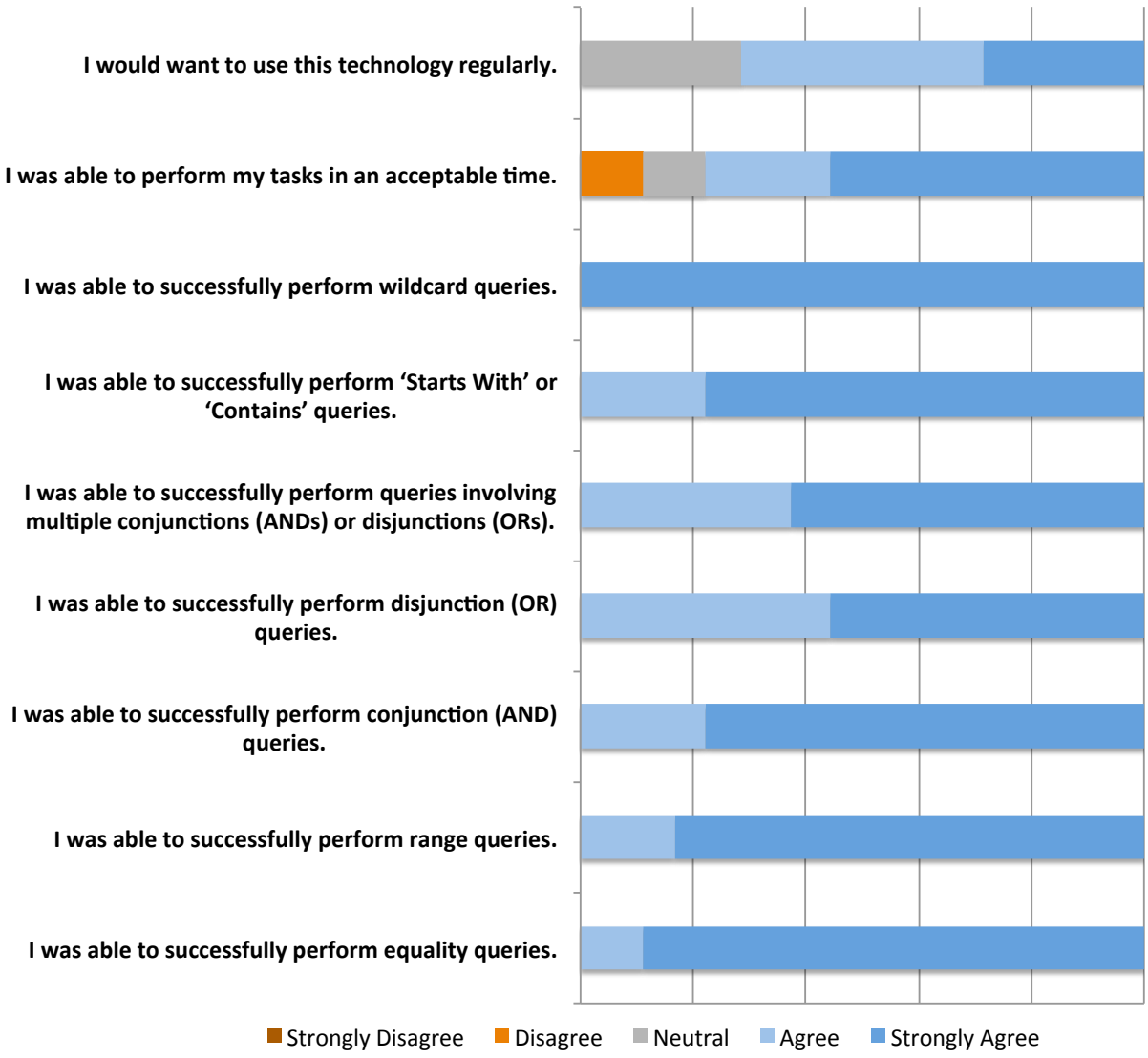


Figure 21: Operator agreement with questions about ESPADA. These questions were administered during the scripted user testing session.

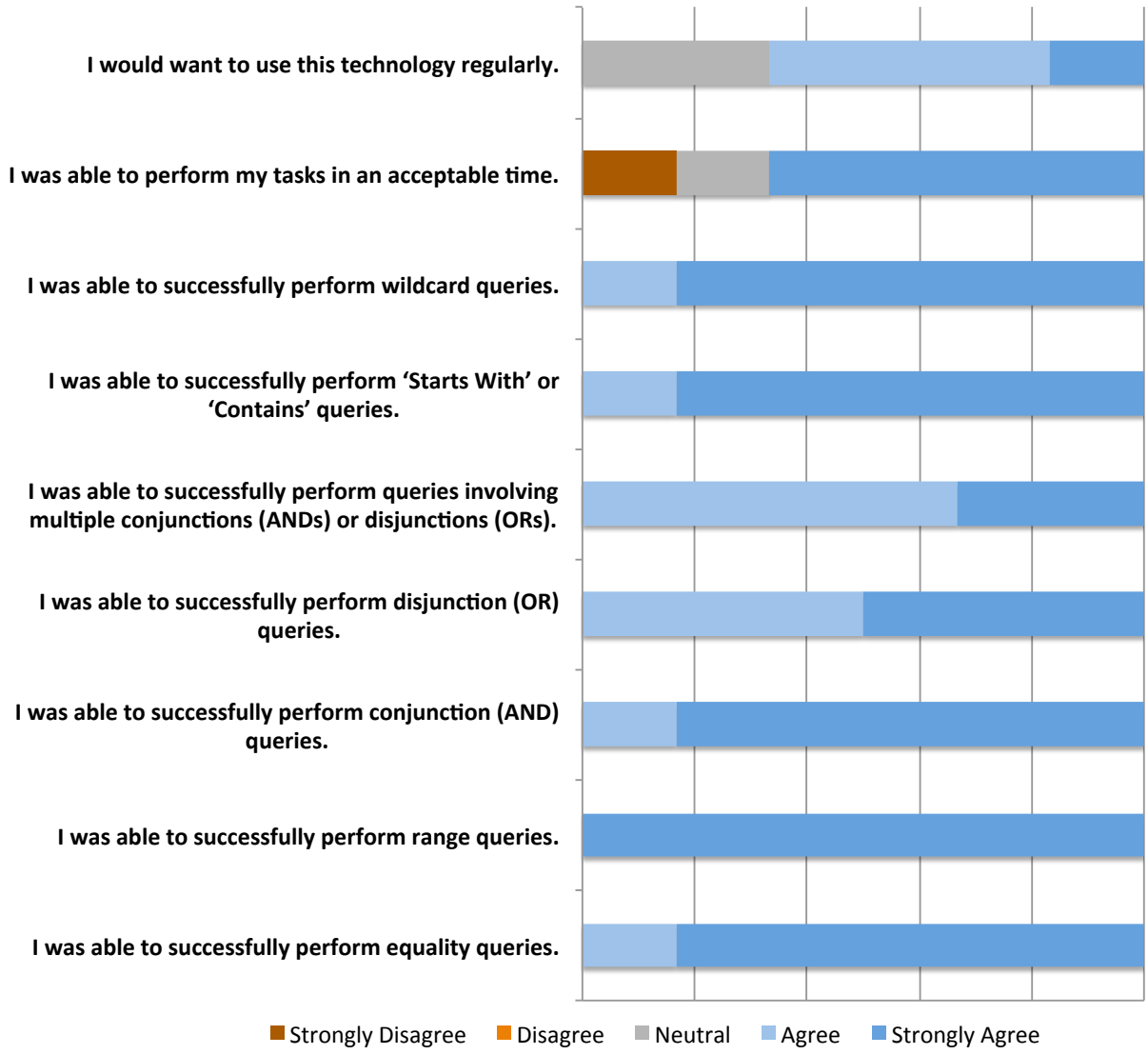


Figure 22: Operator agreement with questions about Stealth. These questions were administered during the scripted user testing session.

References

[1] Intelligence Advanced Research Projects Activity. Intelligence Advanced Research Projects Activity (IARPA) Security and Privacy Assurance Research (SPAR) program sample use cases, 2014.

[2] Gilles Barthe, François Dupressoir, Benjamin Grégoire, César Kunz, Benedikt Schmidt, and Pierre-Yves Strub. Easycrypt: A tutorial. In Alessandro Aldini, Javier Lopez, and Fabio Martinelli, editors, *Foundations of Security Analysis and Design VII 2012/2013 Tutorial Lectures*, volume 8604 of *Lecture Notes in Computer Science*, pages 146–166. Springer, 2013.

[3] Dan Boneh, Craig Gentry, Shai Halevi, Frank Wang, and David J. Wu. Private database queries using somewhat homomorphic encryption. In *Applied Cryptography and Network Security*, volume 7954, pages 102–118. Springer, 2013.

[4] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. In *Innovations in Theoretical Computer Science (ITCS'12)*, 2012. `http://eprint.iacr.org/2011/277`.

[5] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *FOCS*, pages 97–106, 2011.

[6] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *Network and Distributed System Security Symposium (NDSS14)*, 2014.

[7] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology–CRYPTO 2013*, pages 353–373. Springer, 2013.

[8] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *Journal of the ACM (JACM)*, 45(6):965–981, 1998.

[9] Ben Fisch, Binh Vo, Fernando Krell, Abishek Kumarasubramanian, Vladimir Kolesnikov, Tal Malkin, and Steven M Bellovin. Malicious-client security in Blind Seer: A scalable private DBMS. 2015.

[10] Benjamin Fuller, John Darby Mitchell, Robert Cunningham, Uri Blumenthal, Patrick Cable, Ariel Hamlin, Lauren Milechin, Mark Rabe, Nabil Schear, Richard Shay, Mayank Varia, Sophia Yakoubov, and Arkady Yerukhimovich. SPAR pilot evaluation - addendum version 3.0, November 2015.

[11] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `crypto.stanford.edu/craig`.

[12] Craig Gentry and Shai Halevi. Implementing Gentry's fully-homomorphic encryption scheme. In *EUROCRYPT 2011*, volume 6632, pages 129–148. Springer, 2011.

[13] Craig Gentry, Shai Halevi, and Nigel Smart. Fully homomorphic encryption with polylog overhead. In *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 465–482. Springer, 2012. Full version at `http://eprint.iacr.org/2011/566`.

[14] Craig Gentry, Shai Halevi, and Nigel Smart. Homomorphic evaluation of the AES circuit. In *CRYPTO 2012*, volume 7417 of *LNCS*, pages 850–867. Springer, 2012. Full version at `http://eprint.iacr.org/2012/099`.

[15] Craig Gentry, Shai Halevi, and Nigel P. Smart. Better bootstrapping in fully homomorphic encryption. In *Public Key Cryptography - PKC 2012*, volume 7293 of *LNCS*, pages 1–16. Springer, 2012.

[16] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229. ACM, 1987.

[17] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM (JACM)*, 43(3):431–473, 1996.

[18] Shai Halevi and Victor Shoup. HElib. `https://github.com/shaih/HElib`. Accessed: 2014-09-23.

[19] Shai Halevi and Victor Shoup. Algorithms in HElib. In *CRYPTO 2014*, volume 8616, pages 554–571. Springer, 2014.

[20] Ariel Hamlin and Jonathan Herzog. A test-suite generator for database systems. In *IEEE High Performance Extreme Computing Conference*, pages 1–6. IEEE, 2014.

[21] LGS Innovations Columbia University Bell Labs Team. ENTACT pilot preliminary report: BLoom filter INDex SEarch of Encrypted Results (BLIND SEER), 2015.

[22] Intelligence Advanced Research Projects Activity. Broad agency announcement IARPA-BAA-11-01: Security and privacy assurance research (SPAR) program, February 2011.

[23] Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel Rosu, and Michael Steiner. Outsourced symmetric private information retrieval. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 875–888. ACM, 2013.

[24] Rensis Likert. A technique for the measurement of attitudes. *Archives of psychology*, 1932.

[25] MIT Lincoln Laboratory. HEtest. `https://www.ll.mit.edu/mission/cybersec/softwaretools/hetest/hetest.html`, February 2011.

[26] Muhammad Naveed, Seny Kamara, and Charles V. Wright. Inference attacks on property-preserving encrypted databases. In *ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015.

[27] Vasilis Pappas, Fernando Krell, Binh Vo, Vladimir Kolesnikov, Tal Malkin, Seung Geol Choi, Wesley George, Angelos Keromytis, and Steve Bellovin. Blind Seer: A scalable private DBMS. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 359–374. IEEE, 2014.

[28] Raluca A. Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. CryptDB: processing queries on an encrypted database. *Commun. ACM*, 55(9):103–111, 2012.

[29] Project Gutenberg. `http://www.gutenberg.org`.

[30] IBM Research and UC Irvine. IARPA SPAR TA-1 project ESPADA phase 1 final report, March 2013.

[31] IBM Research and UC Irvine. ESPADA phase 2 performance costs, February 2014.

[32] IBM Research and UC Irvine. IARPA SPAR TA-1 project ESPADA final report, June 2014.

[33] IBM Research and UC Irvine. IARPA SPAR TA-1 project ESPADA phase 2 preliminary report, January 2014.

[34] IBM Research and University of California Irvine. IARPA ENTACT project ESPADA preliminary final report, 2015.

[35] Nigel P. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Public Key Cryptography – PKC 2010*, volume 6056 of *LNCS*, pages 420–443, 2010.

[36] Nigel P. Smart and F. Vercauteren. Fully homomorphic SIMD operations. In *Designs, Codes and Cryptography*. Springer, 2011.

[37] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*, pages 44–55. IEEE, 2000.

[38] Stealth Software Technologies Inc. Stealth accomplished goals and response to comments, 2015.

[39] Columbia University Bell Labs Team. BLoom INDex SEarch of Encrypted Results (BLIND SEER), January 2014.

[40] Stealth Software Technologies. SPAR TA1 - Stealth solution - Adversarial model.

[41] Stealth Software Technologies. SPAR TA1 - Stealth solution - Exact performance costs.

[42] Stealth Software Technologies. SPAR TA1 - Stealth solution - Proofs and informal description.

[43] Stealth Software Technologies. Stealth proposed query types for SPAR TA1 – phase 1, 2012.

[44] MIT Lincoln Laboratory Evaluation Team. Security and privacy assurance research (SPAR) program Technical Area 1 evaluation process document: Phase 2 test plan & rules of engagement. Version 1.1, August 2012.

[45] MIT Lincoln Laboratory Evaluation Team. Security and Privacy Assurance Research (SPAR) program Technical Area 1 phase 1 test plan Version 1.3, November 2012.

[46] MIT Lincoln Laboratory Evaluation Team. Security and performance evaluation of Applied Communication Sciences, Boston University, Rutgers University, University of Texas and Yale University phase 1 deliverable SPAR program Technical Area 1 Version 1.1, April 2013.

[47] MIT Lincoln Laboratory Evaluation Team. Security and performance evaluation of Applied Communication Sciences, Spread Concepts, and Rutgers University deliverable SPAR program technical area 3.1 version 1.0, April 2013.

[48] MIT Lincoln Laboratory Evaluation Team. Security and performance evaluation of Argon ST deliverable SPAR program technical area 3.1 version 1.0, April 2013.

[49] MIT Lincoln Laboratory Evaluation Team. Security and performance evaluation of Columbia University and Bell Labs phase 1 deliverable SPAR program Technical Area 1 Version 1.2, April 2013.

[50] MIT Lincoln Laboratory Evaluation Team. Security and performance evaluation of IBM phase 1 deliverable SPAR program technical area 2 version 1.1, May 2013.

[51] MIT Lincoln Laboratory Evaluation Team. Security and performance evaluation of IBM research and UC Irvine phase 1 deliverable SPAR program Technical Area 1 Version 1.1, April 2013.

[52] MIT Lincoln Laboratory Evaluation Team. Security and performance evaluation of Raytheon BBN technologies deliverable SPAR program technical area 3.1 version 1.0, April 2013.

[53] MIT Lincoln Laboratory Evaluation Team. Security and performance evaluation of Stealth phase 1 deliverable SPAR program technical area 2 version 1.0, April 2013.

[54] MIT Lincoln Laboratory Evaluation Team. Security and performance evaluation of Stealth Software Technologies, Inc. phase 1 deliverable SPAR program Technical Area 1 Version 1.1, April 2013.

[55] MIT Lincoln Laboratory Evaluation Team. Security and privacy assurance research (SPAR) program Technical Area 1 evaluation process document: Phase 2 test plan & rules of engagement. Version 1.7, March 2014.

[56] MIT Lincoln Laboratory Evaluation Team. SPAR security and performance evaluation Columbia University and Bell Labs Technical Area 1 deliverable Version 3.1, August 2014.

[57] MIT Lincoln Laboratory Evaluation Team. SPAR security and performance evaluation IBM research and UC Irvine Technical Area 1 deliverable Version 3.1, August 2014.

[58] MIT Lincoln Laboratory Evaluation Team. SPAR security and performance evaluation IBM technical area 2 deliverable version 3.1, August 2014.

[59] US Census Bureau. Census 2000 5-percent public use microdata sample (PUMS) files. `http://www2.census.gov/census_2000/datasets/PUMS/FivePercent/`.

[60] US Census Bureau. Genealogy data: Frequently occurring surnames from census 2000. `http://www.census.gov/genealogy/www/data/2000surnames/`.

[61] Martin van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 24–43. Springer, 2010.

[62] Mayank Varia, Benjamin Price, Nicholas Hwang, Ariel Hamlin, Jonathan Herzog, Jill Poland, Michael Reschly, Sophia Yakoubov, and Robert K. Cunningham. Automated assessment of secure search systems. *Operating Systems Review*, 49(1):22–30, 2015.

[63] Mayank Varia, Sophia Yakoubov, and Yang Yang. HEtest: A homomorphic encryption testing framework. In Michael Brenner, Nicolas Christin, Benjamin Johnson, and Kurt Rohloff, editors, *Financial Cryptography and Data Security - FC 2015 International Workshops, BITCOIN, WAHC, and Wearable, San Juan, Puerto Rico, January 30, 2015, Revised Selected Papers*, volume 8976 of *Lecture Notes in Computer Science*, pages 213–230. Springer, 2015.

[64] Yang Yang. Evaluation of somewhat homomorphic encryption schemes. Master's thesis, Massachusetts Institute of Technology, 2013.

[65] Andrew Chi-Chih Yao. Protocols for secure computations. In *FOCS*, volume 82, pages 160–164, 1982.

A Glossary of Terms

AES Advanced Encryption Standard. Current U.S. standard for symmetric cipher.

API Application Program Interface.

AWS Amazon Web Services

B-tree Balanced tree structure where nodes may have more than two children.

Baseline A non-privacy-preserving database software package (including a standard database server and client executable) produced by the MIT Lincoln Laboratory Evaluation Team. In the original research program Phase 1, the database server was MySQL; Phase 2, will use MariaDB 5.5.32. In the pilot demonstrated, the database server was MySQL.

BLIND SEER SPAR technology developed by Columbia University, Bell Labs, and LGS Innovations.

Bloom filter Space efficient data structure used to represent large sets. Has no false negatives and a configurable false positive rate.

Collusion Two parties P_1 and P_2 are said to *collude* if they communicate outside of the prescribed protocol. For the purposes of security definitions, colluding parties are assumed to be under the control of one entity.

Covert A *covert* party is malicious but has an active interest in not having its malicious behavior be detected.

Database Management System A *database management system* is a software application that allows several clients to query a database. The goal of SPAR Technical Area 1 is for teams to build a privacy-preserving database management system. Examples of existing (non-privacy-preserving) database management systems are MySQL, MariaDB and Oracle.

C2S Commercial Cloud Services. Cloud platform based on Amazon Web Services

Database A *database* is a collection of records, generally organized into tables. Each table is structured by its column and row, such that columns typically indicate structure for the table and rows identify specific entries. The columns are also referred to as fields. All records in a single table have the same set of fields. Current SPAR systems only support a single table. Fields are not fully searchable by default, currently, the set of searchable fields (and query types) must be specified at database creation time.

Data owner Logical entity that is responsible for management of a private data set. Has privacy concerns for the records contained in the data set. The owner can allow a querier access to a record using SPAR protocols. In whatever interactions take place, the owner should not learn the querier's queries or access patterns.

Pilot Data Owner Provided the data set for use during the pilot, specified query policy requirements for access to the data set, and provided operators for the testing period. Details about the Data Owner can be found in [10].

Data querier Logical entity that wishes to ask sensitive queries about a private data set. Has privacy concerns for their query as well as any results to the query. SPAR protocols may support multiple queriers. If protocols support multiple queriers, then they should be able to withstand collusion between multiple queriers. Protocols need not be able to withstand collusion between queriers and the owner.

Pilot Data Querier Specified confidentiality requirements for query privacy, provided the use case and current workflow, and provided operators for the testing period. Details about the Data Querier can be found in [10].

DH Diffie-Hellman. Key agreement algorithm security relies on the hardness of discrete logarithm.

DNF Disjunctive Normal Form. Boolean formula that is an OR of clauses each of which is a conjunctive of simple clauses.

El Gamal Public-key encryption scheme that relies on hardness of discrete logarithm problem.

ESPADA SPAR technology developed by IBM Research and UC-Irvine.

False positive Occurs when a valid query should be accepted but is rejected

False negative Occurs when an invalid query should be rejected but is accepted

Field A *field* or *column* represents a named value in a record. Each field has a data type that defines the types of values that may be associated with the field. All records in the same table have the same fields.

Garbled Circuit Cryptographic approach to allow two parties to jointly and private compute arbitrary circuit.

Homomorphic Encryption Encryption scheme that allows for manipulation of underlying plaintext by manipulatiing only with ciphertext.

Fully Homomorphic Encryption Encryption scheme that allows for arbitrary computation on underlying plaintext by manipulation only ciphertexts.

IARPA Intelligence Advanced Research Projects Activity. Funded the SPAR research program, evaluated the results, and decided on the use case for the pilot demonstration. Identified potential customer organizations within the government, communicated SPAR capabilities to those customers, prioritized technical requirements from those customers, and coordinated the pilot effort.

International Business Machines (IBM) Research - Lead of ESPADA team.[20] This technology was developed under the SPAR program to provide secure database search. Documentation on this technology includes security [30, 32, 33] and performance [31] evaluations by the IBM team and analysis by MIT LL [57].

IPSec Internet Protocol Security. Protocol designed to secure connection between two network devices. Operates at the network layer. Intended to provide confidentiality and integrity.

IT Information Technology.

Lucent Government Systems (LGS) - Led the extension of the BLIND SEER implementation during the pilot demonstration.[21] Throughout this document, this technology is called *BLIND SEER*. This technology was developed during the SPAR research program to provide secure database search. Documentation on this technology includes security and performance evaluations by the Columbia University and Bell Labs team [39] and analysis by MIT LL [49, 56].

Malicious A *malicious* party does not need to follow the prescribed protocol. Additionally, it may passively eavesdrop on network communications between other parties or actively interfere with these communications by inserting, removing, mauling, or replaying communication over the network.

MariaDB Commercial database management system.

MIT LL Testing and evaluation team for the three SPAR technologies for both the research program and the pilot demonstration. MIT LL developed software to facilitate comparisons of the performance and functionality of the three implementations. This included developing components to instrument the systems being tested and to collect metrics on end-to-end performance, as well as writing scripts to manage the lifecycle of systems being tested and for data analysis. Additionally, MIT LL designed tests, analyzed their results, and wrote this report. MIT LL also developed a standard user interface for the pilot demonstration.

MySQL Commercial database management system.

Network The *network* is an implicit third party whose functionality is that of a messenger. An honest network, denoted N, faithfully delivers messages between parties. The network may be malicious; The protocol should address threats to security requirements arising from a compromised network.

OpenSSL Popular opensource cryptographic library written in the C language.

[20] In collaboration with University of California – Irvine

[21] In collaboration with Bell Labs and Columbia University.

Policy Checker A *policy checking algorithm* determines whether or not a client is authorized to make a given query, based on a *query authorization policy* provided by the server. Hence, it protects the server's confidentiality against clients.

Private Information Retrieval Class of cryptographic protocols were a client retrieves an item from a server without the server knowing which item was retrieved.

Pseudorandom function Keyed function whose behavior is indistinguishable from a random function without knowledge of the key.

- **Oblivious Pseudorandom function** A two party protocol for evaluating a pseudoranom function where the key is held by a server and the input by a client. At the end of the protocol the client should learn the pseudorandom function output on the input and no other information should be exchanged.

Query A *query* is a request made by the client to retrieve information from the database. Queries span a range of operations, from returning a single value from a single field to complex operations performed on the data, including data dependencies (such as 'JOIN') or conditional operations. SPAR technology currently supports a subset of common query types.

Query Access Patterns *Query access patterns* refers to any information regarding which records a client is accessing.

RAM Random Access Memory. Data where any location can be efficiently accessed.

- **Oblivious RAM** Cryptographic technique to make all data location accesses data independent. Use to eliminate information leakage due to memory access patterns.

RAID Redundant Array of Independent Disks. Technique to present multiple hardware disk as a smaller number of virtual disks with lower probability of failure.

Record A *record* or *row* is an entry in a table associating fields with values.

REST Representational State Transfer. Architecture for building network applications.

RHEL Red Hat Enterprise Linux.

(Secure) Multi-Party Computation Cryptographic technique for multiple parties to private compute arbitrary function of shared inputs.

Security and Privacy Assurance Research Research program initiated by IARPA. Built tools that balanced security, privacy, and efficiency.

Semi-honest A party is said to be *semi-honest*, or "honest-but-curious", if it obeys the protocol but attempts to glean additional information from messages that it observes.

Semi-honest parties do not eavesdrop on messages between other named parties. Semi-honest behavior permits a partial form of collusion: multiple clients are allowed to collude, but all other types of collusion are forbidden.

SHA-1 Secure Hash Algorithm. Previous U.S. government standard hash algorithm.

SHA-2 Secure Hash Algorithm. Current U.S. government standard hash algorithm.

SHA-3 Secure Hash Algorithm. Current U.S. government standard hash algorithm.

SPAR Abbreviation for Security and Privacy Assurance Research. IARPA sponsored research program on controlled and secure information sharing.

SQL Structured Query Language. Database management language used by many popular database management systems.

SSL Secure Sockets Layer. Used to provide transport layer security between two endpoints. Designed to provide confidentiality and integrity. Replaced by TLS.

Stealth SPAR technology developed by Stealth Software Inc.

Stealth Software Inc. SPAR team that developed the *Stealth* technology.[22] Stealth technology was developed under the SPAR program to provide secure database search. Documentation on this technology includes security [42] and performance [41] evaluations by Stealth and analysis by MIT LL [54].

Symmetric Searchable Encryption Encryption scheme where a client outsources data to an untrusted database server. Though encrypted, the client and database server are able to perform meaningful searches over the data and return the "right" encrypted records to the client.

TA A *technical area* is a research subgoal for a research program. May contain multiple research teams/performers.

Table A *table* is a set of records. Each record has a value in each of the database columns. For the SPAR program it is acceptable if some of the columns are not searchable. The columns are also referred to as fields. All records in a single table have the same set of fields. In the SPAR program all fields in a record will have a non-`NULL` value.

TB Terabyte. Equal to 10^{12} bytes.

Third Party A *third party* is any additional protocol participant, such as an intermediary, escrow agent, compute platform, or key generator. SPAR Protocols may have multiple independent third parties. Third parties should not learn plaintext records, the query authorization policy, or any subset of the ciphertext records in their original order.

[22] During the SPAR research program, this technology was known as FSS. It has changed significantly since the research program and is now known as Stealth.

Value A record associates the set of fields defined for the table in which the record resides with a set of *values*. The value associated with the field must be legal for the data type of the field.

TLS Transport Layer Security. Used to provide transport layer security between two endpoints. Designed to provide confidentiality and integrity. Preceded by SSL.

True positive Occurs when an invalid query is correctly rejected by a query check

True negative Occurs when a valid query is correctly accepted by a query check

XML EXtensible Markup Language.

B Summary of SPAR Technical Areas 2 and 3.1

In this section, we briefly survey the other two Technical Areas of SPAR. TA-2 sought design improvements in somewhat homomorphic encryption, and TA-3.1 desired privacy-preserving publish-subscribe systems. (Note: the other sub-tasks of TA3 listed in the BAA [22] were not funded.)

TA-2: Homomorphic Encryption for Evaluation of Assured Data Retrieval Functions. Homomorphic encryption is a cryptographic primitive that enables computation directly on encrypted data. In 2009, Craig Gentry proposed the first *fully* homomorphic encryption (FHE) scheme that enabled an arbitrary (public) function to be evaluated directly on hidden data [11]. In the six years since Gentry's discovery, there has been substantial research into the design of fully homomorphic encryption algorithms [4, 5, 13, 15, 35, 36, 61]. Unfortunately, these systems have been shown to be quite slow, often 7-9 orders of magnitude slower than computation on unencrypted data [12,14]. One reason for this performance hit is that FHE programs do not operate in the same way that, say, the Intel x86 instruction set does: it has a far more limited set of basic assembly instructions at its disposal, and most programs do not neatly decompose into these instructions.

SPAR TA-2 thus poses a natural question: can we design and implement a *somewhat* homomorphic encryption (SWHE) scheme that sacrifices the ability to compute arbitrary programs in return for dramatically better performance on the programs it does evaluate securely? Specifically, inspired by the goals of the other TAs, performers on TA-2 were asked to build SWHE schemes that efficiency evaluate functions that commonly serve as building blocks in data search systems, such as "binary search, pattern matching, evaluation of predicates, Bloom Filter testing, and computing hash functions [with support for] range matches [and] partial matches" [22]. In principle, such primitives could be judiciously used within the other TAs of SPAR to design even stronger search algorithms. For instance, SWHE would allow the data owner to serve as the database server while retaining the security properties mentioned in Section 1.2.1 above. In particular, the server would not be able to leverage knowledge of the underlying data to learn anything about clients' queries. (We note that this integration between TAs did not actually occur during the SPAR program though.)

Two performer teams participated in SPAR TA-2: a team from IBM Research participated in both phases [3, 18, 19] and a team from Stealth Software, Inc. participated in the first phase. The performers were able to construct SWHE schemes that were "only" 4 to 6 orders of magnitude slower than unprotected computation, and IBM Research demonstrated its application to a small database of 100,000 records in Phase 2. MIT LL has written evaluation reports for both performers [50, 53, 58] and published our evaluation software along with some of our test results [25, 63, 64].

TA-3.1: Privacy Protection in Publish/Subscribe Systems. Similar in nature to TA-1, this Technical Area of SPAR also required performers to design and implement a

secure data search system (with similar assurance requirements to those stated in Section 1.2.1) whose performance is within 10× that of a non-privacy-preserving system. Rather than databases though, this TA focused on publish-subscribe systems that quickly route messages between distributed systems. More concretely, a pub-sub system contains one or more publishers who wish to distribute messages to some subscribers through the aid of a message brokering service.

While conceptually simple, this difference spurred TA-3.1 performers to design technologies that were optimized for vastly different settings:

- Storage vs. streaming: Because databases tend to be large data stores, TA-1 desired performance at large scale (namely, 10 TB). On the other hand, pub-sub systems tend to be used in message queuing applications in which messages are not actually known about beforehand; rather, they are sent on the fly. The pub-sub broker must route them to interested parties faster than the message transmission requests are being sent to it. The pub-sub broker may use a small buffer to store messages that haven't been processed yet, but the amount of storage should be quite small (less than 1 MB).

- Latency vs. throughput: MIT LL's T&E captured latency and throughput metrics for both TAs. But, the focus of the TAs were different: in TA-1, performers were tasked with minimizing the wait time for a client making a single query, whereas in TA-3.1, performers were tasked with maximizing the number of messages that the broker can process without being flooded.

- Number of clients: TA-1's database search problem makes perfect sense with a single client. As a result, the single-client scenario was by far the most common setting, with only 1 performer producing any software that handled the multiple-client scenario. By contrast, the entire purpose of TA-3.1's routing service is to distribute messages among several clients. As such, TA-3.1 systems were required to scale to the level of at least 100 different subscribers, potentially each with a different subscription request made to the broker.

- Query types: As a general principle, the types of subscriptions made in pub-sub systems tend to be much simpler than the types of queries made in SQL databases. The difference requirements for performers reflects this: TA-3.1 performers were only required to support simple equality searches (e.g, `section = "sports"`), although all performers managed to support limited boolean predicates as well.

- Updates: In the database application of TA-1, the data owner was permitted to submit record-level update commands. By contrast, the published messages in TA-3.1 were submitted in a streaming fashion and thus un-updatable by their very nature; instead, the subscribers were permitted to update their interests to the broker (e.g., changing a predicate to `section = "business"`).

Three performers participated in TA-3.1: a team from Argon ST, a team from Raytheon BBN Technologies, and a team from Applied Communication Sciences, Spread Concepts,

and Rutgers University. As per the BAA, this technical area only lasted through Phase 1. MIT LL conducted T&E of all performer systems and provided reports to the government [47, 48, 52].

C SPAR Research Program Queries and Dataset

This section describes, in more detail, the queries and data that were run against the SPAR implementations during testing in the research program. Table 17 describes the schemas used in the SPAR research program.

field name	data type	min size (bytes)	max size (bytes)	min value	max value	number of distinct values
`id`*	int-64	8	8	0	$2^{64}-1$	all distinct
`fname`*	string	2	11			5163
`lname`	string	2	15			88,798
`ssn`*	string	9	9			898,000,000
`dob`*	date	10	10			33,604
`address`	string	20	100			67,541,174
`city`*	string	3	35			17,848
`state`*	enum					52
`zip`*	string	5	5			33,178
`sex`*	enum					2
`race`*	enum					8
`marital_status`*	enum					5
`school_enrolled`*	enum					8
`citizenship`*	enum					5
`income`*	int-32	4	4	-2^{16}	2^{21}	25,438
`military_service`*	enum					5
`language`	enum					97
`hours_worked_per_week`*	uint-8	1	1	0	167	99
`weeks_worked_last_year`*	uint-8	1	1	0	52	53
`last_updated`*	uint-32	4	4	0	$2^{32}-1$	
`foo`*	uint-64	8	8	0	$2^{64}-1$	
`notes1`	string	5000	10,000			
`notes2`	string	500	2000			
`notes3`	string	100	250			
`notes4`	string	20	50			
`xml`	string	100	10,000			
`fingerprint`	blob	60,000	100,000			all distinct

Table 17: 10^5 byte/row database schema (sizes in bytes). Rows with an asterisk (*) also exist in the 10^2 byte/row schema. Notes: (1) data for the final column is based on a database with 10^8 rows, (2) dates were guaranteed to fall between the years 1916 and 2015.

C.1 Query Distribution

The scope of these queries was calibrated based upon on the volume of these queries anticpated to be run over a week-long test, derived from query-response speeds observed during the risk-r1eduction period.

The queries described here were all run in `SELECT id` mode; additionally, one fifth of the queries matching under 10,000 records were repeated in `SELECT *` mode.

All of the queries (both `SELECT id` and `SELECT *`) were run in latency mode, and many of them were repeated in throughput mode. The throughput queries all return between 0 and 100 records, and are split up into test cases by unique combinations of query category, subcategory, and sub-subcategory. Each throughput test case should reference the latency base case for comparison.

A caveat should be applied to the baseline performance of P6 (wildcard) and P7 (subsequence) queries. As implemented in the baseline, these queries required a full table scan of the queried column. This is due to a lack of support for a full text index in the baseline software. If the baseline were supplemented by an engine that supported a full text index, such as Sphinx[23], the performance of these queries would improve.

C.1.1 Equality

Equalities were tested for each database and field combination as described below. The following table describes the number of queries run for each combination of result set size ('r'), database, and field. Note that several field/record size combinations are not possible due to the distribution of our data. Rather than marking a '0' in all such table entries, we leave them blank for legibility.

Table 18: Number of queries run for 'r', database, and field. An * denotes that a field is not present in the 10^2 Byte/row databases.

# records	field	1-10 r	11-100 r	101-1,000 r	1,001 -10,000 r	10,001 -100,000 r
10^5	`fname`	200	100	20		
10^5	`lname`*	200	100	20		
10^5	`ssn`	200				
10^5	`address`	200				
10^5	`city`	200	100			
10^5	`zip`	200	100			
10^5	`dob`	200	100			
10^5	`income`	200	100			
10^5	`foo`	200	100	20		
10^5	`last_updated`	200				

[23] www.sphinxsearch.com

Table 18: Number of queries run for 'r', database, and field. An * denotes that a field is not present in the 10^2 Byte/row databases.

# records	field	1-10 r	11-100 r	101-1,000 r	1,001 -10,000 r	10,001 -100,000 r
10^5	`language`	5	5	2	2	
10^6	`fname`		100	20	10	2
10^6	`lname`*		100	20	10	2
10^6	`ssn`	200				
10^6	`address`	200				
10^6	`city`		100	20		
10^6	`zip`		100	20		
10^6	`dob`		100	20		
10^6	`income`		100	20		
10^6	`foo`	200	100	20	10	2
10^6	`last_updated`	200				
10^8	`fname`				10	2
10^8	`lname`*				10	2
10^8	`ssn`	200				
10^8	`address`	200	100			
10^8	`city`				10	2
10^8	`zip`				10	2
10^8	`dob`				10	2
10^8	`income`				10	2
10^8	`foo`	200	100	20	10	2
10^8	`last_updated`	200				
10^9	`fname`					2
10^9	`lname`*					2
10^9	`ssn`	200				
10^9	`address`	200	100			
10^9	`city`					2
10^9	`zip`					2
10^9	`dob`					2
10^9	`income`					2
10^9	`foo`	200	100	20	10	2
10^9	`last_updated`	200				

Note: Equalities were tested only over one *enum* field (`language`) and database (10^5 `rows`) combination. Equalities were not tested over other enums, as those queries would match large numbers of records and take too long to run. Equalities over fields not tested here

(such as `sex` over the database with 10^9 rows) appear later in P1 (boolean formula) and P8 (threshold) queries.

C.1.2 P1 (boolean formula queries)

We tested two different types of simple boolean queries.

eq-and Conjunctions of equalities were tested over each combination of database and number of conjunctive terms between 2 and 5. Result set sizes ('r') and number of records matching the first term of the conjunction ('ftm', for 'first term matches') for each combination of database and number of conjunctive terms were distributed as per the following table:

	1-10 r	11-100 r	101-1,000 r	1,001-10,000 r
1-10 ftm	10			
11-100 ftm	20	10		
101-1,000 ftm	10	2	2	
1,001-10,000 ftm	2	2	2	≤ 2
10,001-100,000 ftm*	2	2	2	≤ 2

Note: An * denotes an 'ftm' was omitted for the 10^5 record database.

eq-or Disjunctions of equalities were tested for each combination of database and number of disjunctive terms between 2 and 5. Result set sizes ('r') and sums of the number of records matching each term ('stm', for 'sum of term matches') for each combination of database and number of disjunctive terms were distributed as per the following table, where a range of stm values is represented as x - y, with x being the lower stm bound ('lstm') and y being the upper stm bound ('ustm'):

	lstm $<$ r $\leq$ ustm	$\frac{1}{2}$ lstm $<$ r $\leq \frac{1}{2}$ ustm	$\frac{1}{4}$ lstm $<$ r $\leq \frac{1}{4}$ ustm
1-10 stm	10	10	10
11-100 stm	10	10	10
101-1,000 stm	10	10	10
1,001-10,000 stm	10	10	10

Note: for each number of disjunctive terms n, any column where r $< \frac{1}{n}$ stm was not tested.

C.1.3 P2 (range queries)

Lincoln tested two-sided and one-sided inequality queries.

range For two-sided range queries over the `foo` field, the result set sizes ('r') for each database tested were distributed as per the following table for each range size ('rs') bucket $[2^{i-1}+1, 2^i]$ for $i \in [2, 50]$:

	1-10 r	11-100 r	101-1,000 r	1,001-10,000 r	10,001-100,000 r
`foo`	20	10	2	2	2

These queries are intended to demonstrate the dependency of range query response time on range size. Within each range size bucket, the actual range sizes were chosen at random. For two-sided range queries over other ordered fields, the result set sizes ('r') for each database were distributed as per the following table:

	1-10 r	11-100 r	101-1,000 r	1,001-10,000 r
`fname`			10	10
`lname`			10	10
`ssn`	20	50	10	10
`dob`	200	100	20	20
`city`			10	10
`zip`			10	10
`income`			10	10
`last_updated`	100	50	10	10
`language` (only for the 10^5-record database)			10	10

less-than For one-sided less-than inequalities, the results set sizes ('r') for each database were distributed as per the following table:

	1-10 r	11-100 r	101-1,000 r	1,001-10,000 r
`fname`			2	2
`lname`			2	2
`ssn`	100	20	10	2
`dob`	100	20	10	2
`address`	100	20	10	2
`city`			2	2
`zip`			2	2
`income`			2	2
`last_updated`	100	20	10	2
`language` (only for the 10^5-record database)			2	2

greater-than The results set sizes ('r') for each database match those described above for less-than queries. Additionally, we ran the following test:

	1-10 r	11-100 r	101-1,000 r	1,001-10,000 r
`foo`	10	20	10	2

C.1.4 P3 (keyword search queries)

For each database, keyword searches were tested where each query returns approximately 1000 records, and keyword lengths ('kl') were uniformly distributed between 8 and 13:

	8 kl	9 kl	10 kl	11 kl	12 kl	13 kl
`notes4`	10	10	10	10	10	10
`notes3`	10	10	10	10	10	10
`notes2`	2	2	2	2	2	2
`notes1`	2	2	2	2	2	2

Keyword searches with keywords of a fixed length of 9, but with a varying number of matching records ('r') was also tested for each database:

	1-10 r	11-100 r	101-1,000 r	1,001-10,000 r
`notes4`	100	20	10	2
`notes3`	100	20	10	2
`notes2`	20	10	2	2
`notes1`	20	10	2	2

C.1.5 P4 (stemming queries)

Stemming queries were tested in the same manner as the above P3 queries, except that the "keyword length" now refers to the length of the stem in question.

C.1.6 P6 (wildcard search queries)

middle-one Initial wildcard search queries of the form:

```
SELECT id FROM main WHERE field LIKE a_ice
```

where exactly one letter is omitted in the middle of the keyword, were tested with varying keyword lengths (including the wildcard character) but fixed result set sizes of approximately 1000 records.

For each database, the queries were distributed as per the following table:

	8 kl	9 kl	10 kl	11 kl	12 kl	13 kl
city	10	10				
lname	10	10	10	10	10	10
address	10	10	10	10	10	10

Keyword searches were also tested for each database with keywords of fixed length of 9, but with varied numbers of matching records ('r'), as per the following table:

	1-10 r	11-100 r	101-1,000 r	1,001-10,000 r
city	20	10	2	2
lname	20	10	2	2
address	20	10	2	2

Note: `notes` fields were not tested for keyword searches as the keyword would need to be of the same length as the text in the notes field, which was deemed impractical.

C.1.7 P7 (subsequence search queries)

initial Initial subsequence search queries of the form

```
SELECT id FROM main WHERE field LIKE %stuff
```

where only a suffix of the desired values is specified, were tested for each database with varying keyword lengths (including the wildcard character) but fixed result set size of approximately 1000 records.

Queries were distributed as per the following table:

	8 kl	9 kl	10 kl	11 kl	12 kl	13 kl
city	10	10				
lname	10	10	10	10	10	10
address	10	10	10	10	10	10
notes4	10	10	10	10	10	10

We also tested initial subsequence searches for each database with keywords of fixed length of 9, but with varied numbers of matching records ('r'), as per the following table:

	1-10 r	11-100 r	101-1,000 r	1,001-10,000 r
fname	20	10	2	2
lname	20	10	2	2
address	20	10	2	2
notes4	20	10	2	2

both Subsequence search queries of the form `SELECT id FROM main WHERE field LIKE %stuff%`, where only a middle section of the desired values is specified, were tested following the same methodology as P7-initial.

final Final subsequence search queries of the form `SELECT id FROM main WHERE field LIKE stuff%`, where only a prefix of the desired values is specified, were tested following the same methodology as P7-initial.
Note: Each instance of `stuff` should be at least k characters long, where k was specified by individual SPAR teams. By default for Phase 2, we let $k = 4$.

C.1.8 P8 (threshold queries)

Threshold queries were tested over equalities with a fixed n and m but with varying result set sizes ('r') and sums of the numbers of matching records for the first $n - m + 1$ clauses ('sftm') for each database as per the following table:

	1-10 r	11-100 r	101-1,000 r	1,001-10,000 r
1-10 sftm	20			
11-100 sftm	10	2		
101-1,000 sftm	2	2	2	
1,001-10,000 sftm	2	2	2	2
10,001-100,000 sftm*	2	2	2	2

Note: * denotes that a sftm was omitted for the 10^5 record database.

Threshold queries over equalities returning a fixed number of records and fixed stfm but with varying n and m values, were also tested as per the following table:

	$m = 2$	$m = 3$	$m = 4$	$m = 5$
$n = 3$	20			
$n = 4$	20	20		
$n = 5$	10	10	10	
$n = 6$	10	10	10	10

C.1.9 P9 (ranking queries)

ranking A ranking query corresponding to each p8-eq query was run for equalities.

proximity For each database size several alarm-word proximity queries with varying result set sizes ('r') were tested as per the following table:

1-10 r	11-100 r	101-1,000 r	1,001-10,000 r
100	20	10	2

C.1.10 P11 (XML queries)

XML queries were tested with both equalities and ranges. All XML documents had 3 levels, with a fan-out 5 at each level. Since only the leaf nodes have queryable values, all queries were made at the same depth.

Queries were run with both of the following forms:

1. `SELECT id FROM main WHERE xml/node1/node2/leaf_node = x`
2. `SELECT id FROM main WHERE xml//leaf_node = x`

Result set sizes for the query formats specified above were distributed as follows:

1-10 r	11-100 r	101-1,000 r	1,001-10,000 r
100	20	10	2

D Risk Reduction Environment

This section describes the risk reduction environment used in the pilot demonstration (see Section 3). This environment was an unclassified cloud-based based on OpenStack.[24] The purpose of this environment was for MIT LL personnel to deploy SPAR implementations and gain experience with the software and processes before transferring to the classified pilot environment. The risk reduction environment included seven (7) compute nodes, each with 256GB of RAM, two (2) Intel E5-2650v3 processors, four (4) 480 GB Solid State Drives, and four (4) 4.0TB SATA drives. Each machine had a 10-gigabit Ethernet connection to a switch in the same rack. The machines each ran Piston OpenStack, a commercially supported OpenStack distribution. For two weeks prior to the formal risk reduction period, SPAR teams were given access to a web interface, which allowed them to create and destroy virtual machine "instances" where the work was performed. The goal of this period was for SPAR teams to work with MIT LL to develop procedures for rapid deployment of their software.

The virtual machine instances were set up using Vagrant[25] version 1.7.2. This allowed new instances of the machines to be spun up at will. To initialize each virtual machine a custom install script was run. This included initializing network settings, limiting ports in use, mounting external drives, and SPAR implementation specific installations. To make the test as similar to the pilot environment, dependencies were pulled offline to form a local repository. There were two virtual machines automatically created for each SPAR implementation; a webserver loaded with MIT LL's web interface and an unprotected database used for comparing results. Each SPAR implementation, depending on their architecture, required additional machines to run their data owner, data querier, database server, and other components, such as a query checker. Each required their own specific vagrant setup file. For example, whenever a new database machine was started, the MySQL database was re-ingested as part of the vagrant process.

D.1 Risk Reduction Dataset

The data used in the risk reduction was created by MIT LL. The goal was two-fold: 1) to contain similar data characteristics to the pilot dataset 2) to have a theme for discussion with SPAR teams. MIT LL choose an insurance company database as the them. It included information about individuals, their policies, and reported accidents. An attempt was made to match the number and type of fields, as well as the distribution, of the columns that would be used in the pilot. All told there were 41 fields consisting of a mix of strings, dates, and integers. The full schema can be seen in table 19. SPAR teams were asked to support equality (EQ), boolean (P1), ranges (P2), wildcards (P6) and subsequence (P7) across specific columns, both with known and unknown universe size.

[24] https://www.openstack.org/
[25] https://www.vagrantup.com/

Table 19: Risk Reduction Schema

name	type	min. size	max size	min value	max value	# of values
ID	bigint			0	$2^{64}-1$	10^8
Title	string	0	15			
FirstName	string	0	20			
MiddleName	string	0	20			
LastName	string	0	35			
Suffix	string	0	8			
Gender	string	0	3			26
DOB	date			1/1/1900	6/1/2015	
Citizenship	string	0	3			
BuildingNumber	string	0	6			
StreetName	string	0	35			
ApartmentNumber	string	0	8			
CityName	string	0	70			
PostalCode	string	0	9			30000
StateCode	string	0	2			50
Country	string	0	3			700
PolicyExpiration	date			1/1/2010	12/31/2020	
InsuranceCompany	string	0	5			
PolicyName	string	0	28			
PolicyNumber	string	0	7			
PolicyZip	string	0	5			30000
PolicyCountry	string	0	3			700
AccidentType	string	0	1			
ClaimNumber	string	0	14			
DriverType	string	0	3			10
AccidentTime	datetime			1/1/1900	6/1/2015	
DriverLocation	string	0	5			30000
AtFaultLocation	string	0	5			30000
InsuranceCoverage	string	0	5			30000
PoliceDriverLocation	string	0	5			30000
PoliceAtFaultLocation	string	0	5			30000
PoliceInsuranceCoverage	string	0	5			30000
Code	string	0	35			
LicenseType	string	0	3			
LicenseNumber	string	0	9			9000000000
ExpirationDate	date			1/1/2000	12/31/2020	
IssuanceDate	date			1/1/1990	12/31/2025	
IssuanceCountry	string	0	3			700
A	tinyint			0	1	2
B	tinyint			0	1	2

Table 19: Risk Reduction Schema

name	type	min. size	max size	min value	max value	# of values
C	tinyint			0	1	2

E Comparison of SPAR Research Tests and SPAR Pilot Tests

The body of this document presents the SPAR pilot demonstration. Prior to the pilot demonstration, MIT LL conducted the SPAR research tests [46, 49, 51, 54, 56, 57]. Both studies were conducted to evaluate the state of SPAR technology. The primary objective of the previous SPAR research tests was an initial evaluation to measure the speed and accuracy of SPAR technology. Whereas, the primary goal of the pilot demonstration presented in this document was to observe the usability of SPAR technology by operators and determine the readiness of SPAR technology for a deployment. These related but differing objectives led to differences in the two studies. This appendix highlights their most salient differences.

Testing Methodology

1. Query Performance: The research testing evaluated the speed of executing a large number of different queries, often with minor variations. The SPAR implementations were required to perform within a reasonable factor against a non-privacy-preserving database instance. The research testing also included testing for query throughput. On the other hand, the pilot demonstration focused more narrowly on timing performance in two areas: the effect of inserts on subsequent query performance and how query-response performance affected the perceptions of live human operators.

2. Response-Accuracy Performance: Both the research testing and the pilot demonstration were concerned with ensuring the accuracy of SPAR queries. The research testing evaluated the correctness of a large number of queries the entire contents of the returned row. Queries in the pilot demonstration were also evaluated for correctness by running the same queries in parallel on a standard MySQL server but only evaluated the presence of a record rather the content of the record itself.

3. Database Modifications: The research testing's modification evaluation considered the atomicity of small-scale database inserts and deletes. The pilot demonstration's modification evaluation focused more on insert rate of the database and timing of queries after up to 500 thousand rows were inserted into the database.

4. Tested Queries: SPAR research testing used on the order of 10^4 synthetic queries. These included ranking, XML, and m-of-n queries. The pilot demonstration focused on about 200 generated by live operators. These queries tended to include more equalities, ranges, substrings, wildcards, and complex boolean conjunctions.

5. Testing Database: The research test used entirely synthetic data with all fields populated and with a large number of field types. It included a billion rows and 20 fields such as text fields up to 10kB in size, enumerated values, binary blobs, and XML fields. All but one of these fields were searchable. In contrast, the pilot demonstration used both real-world data and synthetic data designed to have a similar distribution

to the real-world data. The database was more sparse and smaller, with 10 million rows and many null values. The pilot demonstration database contained an order of magnitude more fields than that of the research test, with about a third being marked as searchable.

6. Query-Checking Correctness: Query-check rules specify which queries are permitted on a given database. The research tests evaluated the accuracy of query-check rules designed to capture a wide range of possible policies. In the pilot demonstration, query checking was extensively tested for a single set of query-check rules based on a real-world use case. This is consistent with the pilot demonstration's greater focus on real-world deployment.

Testing Environment The SPAR research tests look place on a single-purpose server with dedicated machines that had nothing else running on them. These machines ran Ubuntu 12.04 LTS. The SPAR implementations were provided as standalone binaries. These implementations were controlled by a testing harness and communicated through standardized input/output protocol. The network used in the research tests contained an isolated gigabit Ethernet LAN using a switch reserved solely for use by SPAR. This more isolated setup was derived from the research testing's focus on fine-grained performance metrics.

In contrast, the pilot demonstration took place in a real-world setting. The pilot machines used CentOS 6 deployed in C2S. The SPAR implementations were each provided as a set of adapter libraries that linked into the testing framework directly. The pilot environment was not isolated and may have included other traffic. Operator testing occurred at location utilizing web browsers to connect to the cloud environment.

F SPAR Technology Pilot Results

F.1 BLIND SEER

Columbia University, Bell Labs, and LGS jointly developed the BLIND SEER implementation. This section provides detailed results on the performance of BLIND SEER during the pilot test. This material supports the information found in Sections 4,5,6, 7 and 8. This material has been communicated to the BLIND SEER team and incorporated into their pilot report [21].

F.1.1 Pilot Extensions

BLIND SEER added several features during the pilot period. These features were based on the pilot requirements document written by MIT LL (Appendix **??**) as well as discussions with MIT LL personnel while preparing for the pilot. Security implications of these new features have not been evaluated by MIT LL. The BLIND SEER report discusses the implementation of these features [21]. These new features include:

- Implementation of substring and wildcard query types.
- Support for NULL fields.
- Configuration for query-check policy.
- Implementation of MIT LL programmatic API.
- Extension of policy checking to support rules described in Section 3.2.6.

F.1.2 Deployment Experience

This section is based on MIT LL's experience deploying BLIND SEER and is subjective. The technology was somewhat challenging to setup and configure, but issues were resolved thanks to timely support from the LGS team. Some of these issues are likely to be merely a function of prioritizing the development work.

Installation The actual build of software for BLIND SEER was relatively straightforward. The procedure called for a separate build machine that was used only for compiling the software. The MIT LL team chose to build the software on one of the machines used to host BLIND SEER components.

Configuration The configuration of the LGS system appeared complicated and error-prone. Ingest ran fairly quickly and was straightforward with some exceptions. Part of the ingest process involved manually copying a generated schema file to two different places on each of the four machines hosting BLIND SEER components. This was problematic when ingest had to be re-done due to issues in parsing the annotations file, and these replicated

schema files became out of sync. There was no consistency checking of these replicated files, and no feedback in the error logs indicating any sort of inconsistency. The inconsistency was discovered only through time-consuming troubleshooting of erratic and unexpected system behavior. In addition, there was a defect in policy configuration that wasn't discovered until the pilot demonstration; the BLIND SEER team provided a code drop that resolved the issue.

Operation A major difficulty with using the BLIND SEER system was that its indexing service took about 15 to 20 minutes to start up. This was particularly problematic when coupled with the worst-case query times and the relatively short (30 minute) sessions with operators during the pilot demonstration. It meant that if an operator entered a query that took a long time, the MIT LL team had to make tradeoffs between letting the query run to completion and having the system ready to receive queries for the *next* operator session.

Several queries failed during automated execution of the query corpus. Given the long processing time required to run the entire query corpus, there was not sufficient time to re-run these queries once the failure was discovered. Approximately 40 queries are excluded from post-insert performance. Therefore the post-insert performance metrics are not representative. Some of the queries that led to the worst pre-insert query performance could not be incorporated into the post-insert performance measurement.

BLIND SEER included the capability to set logging verbosity at startup time to facilitate troubleshooting. The logging output provided by BLIND SEER components was very helpful in troubleshooting, diagnosing, and resolving issues.

The connectivity of BLIND SEER components seemed the most resilient of the implementations. Because their indexing service required a long time to start up (around 15-20 minutes), they had made it very tolerant of disconnections by peer components.

F.1.3 Pilot Performance Results

How quickly BLIND SEER responded to queries depended heavily on the entropy of data fields and network characteristics. BLIND SEER had a very large variability in query-response time, with some queries taking a very long time to execute as seen in Figure 23. This variability made it difficult for operators to use the system because they did not know when queries would complete. The duplication of records described in Section 3.2.2 may have negatively affected the performance of the BLIND SEER implementation, because its performance improves as database entropy increases. BLIND SEER had much better performance in the risk-reduction environment, though the cause of this is not known. The MIT LL programmatic API had a function to indicate whether a particular query was supported by the implementation. Any query that could not be parsed or answered should have returned *not supported*. Several queries returned inaccurate results or errors (Table 21):

- There were several queries with length 3 substrings. These queries were not supported by BLIND SEER, but the software did not report this and returned incorrect results on these queries.

	Technology	Overall Mean	Standard Deviation
Pre-Insert	BLIND SEER	1470	4770
	MySQL	2.01	6.54
Post-Insert	BLIND SEER	1120	2750
	MySQL	2.10	6.61

Table 20: BLIND SEER Performance Summary, times in seconds

Queries Answered	Accurately	Inaccurately	Error
BLIND SEER	156	17	29
MySQL	202	0	0

Table 21: BLIND SEER Query Expressivity, number of queries. Accurate queries returned without error and with the correct results. Inaccurate queries returned without error but did not match the MySQL baseline. Error queries include queries reported as not supported as queries that returned an explicit error.

- The "_" character was erroneously treated as a wildcard character when searching for exact equality.
- Parsing was limited and depended on spaces between the field and value.

Inserts were processed as they arrived, which was fairly quickly, as seen in table 22. Due to how BLIND SEER was configured when it was set up, MIT LL was not able to run the one million inserts on top of the other inserts. The structure was created with only a fixed number of empty spaces to accommodate inserts. Deletes were processed as they arrived and were handled quickly, taking approximately .0056 seconds per records (see Table 23).

F.1.4 Policy Testing

BLIND SEER does correctly check for the inclusion of a Boolean field, that the Boolean field has the correct value. It also checks that a date range is a subset of the allowed range. MIT LL expected BLIND SEER to be able to handle correctly formed queries in disjunctive normal form (DNF). Unfortunately, this feature appears not to be working. This meant that BLIND SEER incorrectly rejected all valid DNF queries as seen in table 24. BLIND SEER did successfully reject all invalid conjunctive queries. While BLIND SEER also correctly rejected invalid DNF queries, this may have been just because they were DNF as seen in table 25. The BLIND SEER team [21] claims the ability to support queries in DNF format

Records Inserted	10,000	100,000	500,000	1,000,000
Time (s)	1,360	11,600	58,200	-

Table 22: BLIND SEER Insert Performance, time in seconds

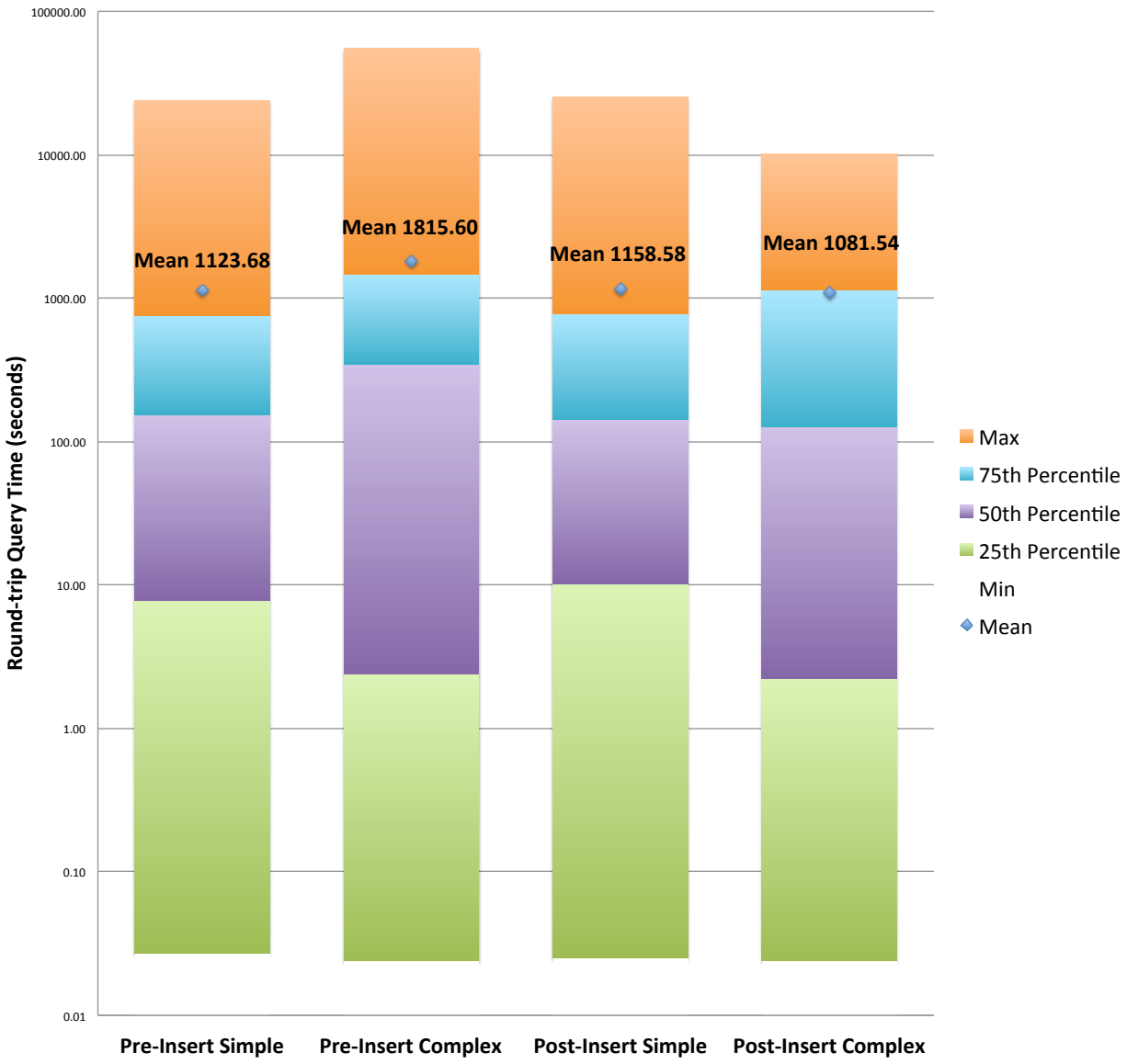


Figure 23: BLIND SEER Pilot Performance. Query response times are split by simple/complex queries and pre/post-insert performance.

Query	Records Deleted	Time(s)
SELECT * FROM base WHERE LastName = ‘Rabe’;	119	1
SELECT * FROM base WHERE FirstName = ‘Sunny’;	363	2
SELECT * FROM base WHERE FirstName = ‘Hunter’;	562	3
SELECT * FROM base WHERE FirstName = ‘Will’;	1022	6
SELECT * FROM base WHERE FirstName = ‘Doug’;	2181	12
SELECT * FROM base WHERE FirstName = ‘Luke’;	2237	14
SELECT * FROM base WHERE LastName = ‘Fuller’;	2727	15
SELECT * FROM base WHERE FirstName = ‘Max’;	3200	17
SELECT * FROM base WHERE FirstName = ‘Karl’;	3948	22
SELECT * FROM base WHERE FirstName = ‘Ruben’;	4538	24

Table 23: BLIND SEER Delete Times in Seconds. All queries were executed after delete to ensure all records were deleted.

but this has not been evaluated by MIT LL. Overall, the ability to enforce the desired policy was high as seen in table 26.

Error rates for valid queries	**BLIND SEER**
Valid Conjunctive Queries (63)	0%
Valid DNF Queries (101)	100%

Table 24: Error rates for valid queries. Lower numbers are better.

Error rates for invalid queries	**BLIND SEER**
Boolean Field Omitted (167)	0%
Boolean Field Wrong Value (242)	0%
Range Wrong Value (195)	0%
Range Too Big (141)	0%

Table 25: Error rates for invalid queries. Lower numbers are better.

F.1.5 Operator Feedback

In this section we provide usability feedback from operators during the pilot demonstration. Most of the feedback on the BLIND SEER system focused on its poor worst-case query-response times. Agreement with Likert questions [24] about usability for BLIND SEER is depicted in Figure 20. Some operators were only able to issue two queries during their 30 minute session. Operators expected the performance of the system to decrease as queries were more complicated. The BLIND SEER system proved slightly counterintuitive as its performance is based on the most selective search term. Thus, complicated queries can actually have better performance. The following operator quotes exemplified their sentiment.

Ability to support Tested Business Rules	BLIND SEER
Rule 0 (Allowing Valid Queries)	Low
Rule 1 (Inclusion of Boolean Value Set to 0)	High
Rule 2 (Inclusion of Date Set to Proper Range)	High
Rule 3 (Restriction of Date to Small Range)	High

Table 26: Current ability to enforce Business Rules (see 3.2.6). Based on the mapping between Tables 24 and 25 and the list of Business Rules.

- "Simple [field1 equality AND field2 equality] query took a long time."
- "Common searches were very slow."
- "The query took longer than expected."
- "This provides a capability I don't have. If it's the fastest thing out there, then I would use it and account for the slowness in staffing/schedule. However, if something faster is available, I'd migrate to that."
- "If faster, results could provide a good amount of information and is easy to use."

F.1.6 Identified Limitations

The following limitations in the technology were observed by the MIT LL team:

1. Substrings of length three were not properly rejected as not supported. This resulted in that term not being considered in query processing and incomplete results being returned. The LGS team has indicated that this is a parsing issue and not an encryption issue, and that this could be addressed [21].
2. Query parsing failed when there was no space between fields/values and operators (e.g. "field = value" works, but "field=value" does not). The LGS team has suggested that a better parser could be developed with a "moderate level of effort" [21].
3. Slow startup (15-20 min) of index service – This was a problem when the system had to be restarted either because a failure occurred during operator testing or when a query was taking a long time to process. This became a real problem for the MIT LL team when coupled with the large variability in query-response time. The LGS team has indicated that since the pilot demonstration, they have implemented a new, faster methodology for startup [21].
4. Extreme variability in response time (6 orders of magnitude) was very problematic for operators. It made it difficult for them to integrate the system into their workflow, because they could not determine whether to wait for a response or move on to another task while the system processed the query. Query-response times were often greater

than 1 hour in those cases. The BLIND SEER team has indicated that they have already made improvements to query speed after the pilot demonstration. They say that that they have already made queries "5 to 20 times faster." They discuss further improvements, and conclude that speeds "can be improved by at least 20x consistently across queries of varying sizes, and likely up to 100x" [21].

5. Date ranges where the start date was later than the end date were not handled correctly. The correct response should be to return zero records. Current behavior is returning "not supported."

6. In at least one case a query clause containing Field = "*v_lue*" was treated as having a wildcard. Wildcards should be of the form Field LIKE "*v_lue*" and the clause above should be treated as equality with a literal string that just happens to contain an underscore.

F.2 ESPADA

IBM Research and University of California - Irvine jointly developed the ESPADA implementation. This section provides detailed results on the performance of ESPADA during the pilot test. This material supports the information found in Sections 4,5,6,7 and 8. This material has been communicated to the ESPADA team and incorporated into their pilot report [34].

F.2.1 Pilot Extensions

The ESPADA team added several features during the pilot period. These features were based on the pilot requirements document written by MIT LL (Appendix **??**) as well as discussions with MIT LL personnel while preparing for the pilot. Security implications of these new features have not been evaluated by MIT LL. The ESPADA report discusses the implementation of these features [34]. These new features include:

- Implementation of basic query reordering for improved performance, since ESPADA's performance is dependent on the order of clauses in the submitted query.
- Wildcard allowed to be one of first two characters in string (still cannot be one of last two characters).
- Increased schema flexibility, including support for NULL fields.
- Switch to allow public policy check result.
- Implementation of MIT LL programmatic API.
- Support for all query types on update data.
- Increased support for subsequence queries.
- Extended range queries to support all data types.

F.2.2 Deployment Experience

This section is based on MIT LL's experience deploying ESPADA and is subjective. The technology was somewhat challenging to setup and configure, but issues were resolved thanks to timely support from the IBM team. Some of these issues are likely to be merely a function of prioritizing the development work.

Installation The documentation for ESPADA was complex. There was a lot of superfluous information that was not really necessary for deployment of the software. The actual build of software for ESPADA was relatively straightforward. The MIT/LL team had to modify some of the build configuration to get the code to compile correctly.

Configuration ESPADA has an extensive list of configurable parameters. However, this expressivity was unnecessary for the pilot demonstration. There did not seem to be a simple configuration process when this expressivity was not necessary. This made the documentation seem complex and verbose. There were some problems during configuration and some concerns over ingest. The ESPADA ingest process was known to take a long time. When the processing completed much earlier than expected there was some uncertainty as to whether the process had completed successfully. The MIT/LL team was dubious about starting the system without verifying that ingest had completed correctly, in part because of the amount of time required to re-perform ingest. The ESPADA team provided timely support, identifying where to look for indications of ingest failure and the MIT/LL team eventually was satisfied that ingest completed correctly.

After ingest, the documentation advocated a procedure involving mounting and unmounting a drive to make the encrypted database available on the third party machine. While this was fairly straightforward in our cloud hosting environment, it would have been rather awkward in a more traditional enterprise setting. The policy specification for ESPADA is complex. ESPADA's policy enforcement specifies exact forms of allowed queries. Enforcing the tested business rules (see Section 3.2.6) required over 600 lines of ESPADA's enforcement language.

Operation There was at least one occasion where the system crashed during automated performance measurement before all queries had been run. The system was restarted and the remaining queries were re-run without incident to complete the measurements. Insert processing for 500k inserts took so long that the MIT/LL team did not have sufficient time remaining to insert 1M records and still have time to complete post-insert performance measurements.

After updates had been applied the MIT/LL team observed that there was a significant increase in startup time for the ESPADA services. This is a result of the synchronization protocol performed between the server and third party at startup to guarantee consistency of inserted records in case of a crash. It illustrates that the ESPADA team has considered what failure scenarios might exist and how the system should behave to guarantee consistency under those conditions. However, the fact that, after any updates have been applied, this protocol is performed *every time* the processes are restarted seems unnecessarily inefficient and time consuming. It took at least 20 minutes to complete this synchronization protocol, and only then were the services available to accept query requests.

The ESPADA build system included the capability to recompile the components with a specified logging verbosity to facilitate troubleshooting. The logging output provided by ESPADA components was very helpful in troubleshooting, diagnosing and resolving issues.

When starting up the system, the ESPADA components can be started in any order, which was straightforward and convenient. ESPADA's third party "services" did not shut themselves down automatically when a peer disconnected, which is probably the correct behavior. There was a connectivity bug that the ESPADA team identified while the pilot demonstration was in progress. This could have been the root cause of the crash mentioned

	Technology	Overall Mean	Standard Deviation
Pre-Insert	ESPADA	97.8	352
	MySQL	2.01	6.54
Post-Insert	ESPADA	108	383
	MySQL	2.10	6.61

Table 27: ESPADA Performance Summary, time in seconds

Queries Answered	Accurately	Inaccurately	Error
ESPADA	190	5	7
MySQL	202	0	0

Table 28: ESPADA Query Expressivity, number of queries. Accurate queries returned without error and with the correct results. Inaccurate queries returned without error but did not match the MySQL baseline. Error queries include queries reported as not supported as queries that returned an explicit error.

previously during performance measurement. The ESPADA team developed a patch, but the MIT/LL team decided that, as late as it was in the pilot demonstration schedule, a code drop was not necessary.

F.2.3 Pilot Performance Results

ESPADA quickly responded to queries in a specific form called searchable normal form (see [30–33, 51, 57]), which requires queries to have the form of a conjunction, where the first term is an equality. For queries presented in this form, ESPADA can be very fast, occasionally outperforming the MySQL baseline implementation. However, response times were much slower for queries not in this form. This led to ESPADA having significant variation in query-response times as shown in Figure 24. There were several problems that led to inaccurate results or errors (see Table 28):

- Inclusion of multiple wildcard selectors in a single query.
- Substrings of length 3 occasionally were not always searched, leading to incomplete results.
- The wildcard character could not be in the last character.
- Quotes around integer values were not parsed.

The ESPADA system had strong atomicity properties (on a per record basis), resulting in a slower update rate (see Table 29). Deletes were processed as they arrived and were done quickly, taking approximately 0.12 seconds per records (data in Table 30).

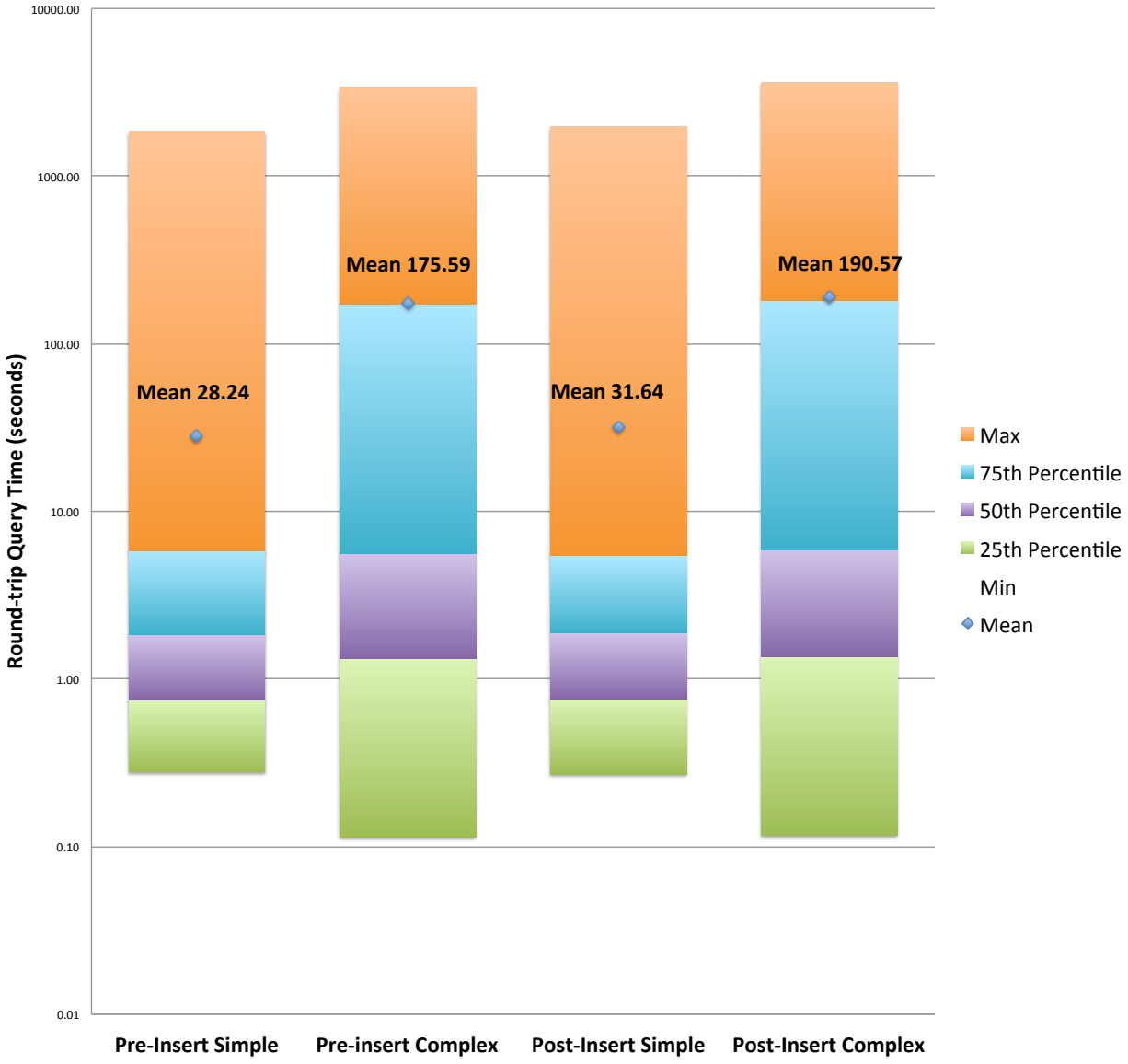


Figure 24: ESPADA Pilot Performance. Query response times are split by simple/complex queries and pre/post-insert performance.

Records Inserted	10,000	100,000	500,000	1,000,000
Time (s)	2,880	28,300	137,000	-

Table 29: ESPADA Insert Performance, time in seconds

Query	Records Deleted	Time(s)
SELECT * FROM base WHERE FirstName = 'Eric';	294	36
SELECT * FROM base WHERE FirstName = 'Scott';	297	37
SELECT * FROM base WHERE FirstName = 'Gary';	328	41
SELECT * FROM base WHERE FirstName = 'Kevin';	357	44
SELECT * FROM base WHERE FirstName = 'Daniel';	499	63
SELECT * FROM base WHERE FirstName = 'Mark';	526	64
SELECT * FROM base WHERE FirstName = 'Mary';	1427	178
SELECT * FROM base WHERE FirstName = 'Robert';	1718	216
SELECT * FROM base WHERE FirstName = 'John';	1801	229
SELECT * FROM base WHERE Citizenship = 'USA';	18708	2280

Table 30: ESPADA Delete Times in Seconds. All queries were executed after delete to ensure all records were deleted.

Error rates for valid queries	**ESPADA**
Valid Conjunctive Queries (63)	3%
Valid DNF Queries (101)	63%

Table 31: Error rates for valid queries (lower numbers are better).

F.2.4 Policy Testing

ESPADA successfully enforces query structure and range size, but does leak this information to the server. In addition to the searchable normal form mentioned previously, ESPADA also supports queries in disjunctive normal form (DNF). However, ESPADA incorrectly rejected many valid DNF queries as seen in table 31. Further, with the exception of validating range size, ESPADA was unable to check for specific values in a query as seen in table 32. For example, it could check whether a query looked at column A, but could not check whether the query ensured A had a value of false. A further difficulty is that ESPADA has a very complex system for setting up its query check rules. The query check rules presented here became over 600 individual rules in ESPADA, requiring an explicit declaration of all possible query structures. Overall, the ability to enforce the desired policy was low as seen in table 33.

Error rates for invalid queries	**ESPADA**
Boolean Field Omitted (167)	0%
Boolean Field Wrong Value (242)	48%
Range Wrong Value (195)	61%
Range Too Big (141)	0%

Table 32: Error rates for invalid queries (lower numbers are better).

Ability to support Tested Business Rules	ESPADA
Rule 0 (Allowing Valid Queries)	Low
Rule 1 (Inclusion of Boolean Value Set to 0)	Low
Rule 2 (Inclusion of Date Set to Proper Range)	Low
Rule 3 (Restriction of Date to Small Range)	High

Table 33: Current ability to enforce Business Rules (see 3.2.6). Based on the mapping between Tables 31 and 32 and the list of Business Rules.

F.2.5 Operator Feedback

In this section we discuss the usability of ESPADA during the pilot demonstration. Figure 21 depicts agreement with questions about using ESPADA asked via questionnaire during the "scripted" testing session. The ESPADA system had wide variation in performance. For queries of the right form, the system returned very quickly and operators found the system very responsive. Wildcards and queries without a leading conjunction were significantly slower. Operators expected performance to scale with the number of records returned. This is not the primary factor for determining the query time for ESPADA. This feedback is reflected in the quotes below:

- "This provides a capability I don't have. If it's the fastest thing out there, then I would use it and account for the slowness in staffing/schedule. However, if something faster is available, I'd migrate to that."
- "If accurate results can be returned consistently, I can see an obvious advantage over other databases."
- "This technology is easy to use and with the addition of applications it could allow for a central, encrypted, hub of information."
- "The last query took a long time but considering the number of results this is to be expected."
- "Wildcard seemed to be significantly slower than other searched. [sic]"
- "Seems to work quickly and effectively."
- "It's always useful for analysts to have direct access to data sets that will benefit their work. If the technology is able to streamline the process of obtaining that data, analysts will find applications for it."

F.2.6 Identified Limitations

The following limitations in the technology were observed by the MIT LL team:

- Small substrings of length three were not properly identified as unsupported. Current behavior is omitting searching on these records.
- Quotes around integer values were not supported (e.g. WHERE Field = "1")
- Significant variability in response time (several orders of magnitude) was very problematic for operators. It made it difficult for operators to integrate the system into their workflow, because they could never predict whether to wait for a response or move on to another task while the system processed the query.
- Unable to enforce policy by value.
- Can't include wildcard as the last character.
- Date ranges where the start date was later than the end date were not handled correctly. The correct response should be to return zero records. Current behavior is returning not supported.

F.3 Stealth

Stealth Software Inc. developed Stealth technlogy.[26] This section provides detailed results on the performance of Stealth during the pilot test. This material supports the information found in Sections 4,5,6, 7, and 8. This material has been communicated to the ESPADA team and incorporated into their pilot report [38].

F.3.1 Pilot Extensions

The Stealth team added several features during the pilot period. These features were based on the pilot requirements document written by MIT LL (Appendix **??**) as well as discussions with MIT LL personnel while preparing for the pilot. Security implications of these new features have not been evaluated by MIT LL. The main feature added during the pilot was support for Boolean queries. The Stealth report [38] contains a high level description of this extension. This description sounds plausible; however, it is not detailed enough for a thorough evaluation. Before any production use of Stealth, this extension must be rigorously evaluated. Note that this implementation supports conjunctions and disjunctions; negations are not currently supported. Information on these extensions can be found in the Stealth pilot report [38]. Other features added during pilot period:

- Support for wildcard character queries.
- Support for NULL fields.
- Initial support for query structure policy enforcement.
- Configuration for public policy check result.
- Implementation of MIT LL programmatic API.
- Support for inserted data records after initial ingest.

F.3.2 Deployment Experience

This section is based on MIT LL's experience deploying Stealth and is subjective. The technology was easy to set up, configure and maintain. Overall, the system was robust and testing was able to proceed smoothly and without incident. In the following subsections, we observe a number of issues from our experience deploying the Stealth software. These concerns are a reflection of the maturity of the software and a deliberate choice by the Stealth team to prioritize more essential features in time for the pilot demonstration. Resolving these concerns seems relatively straightforward, and is most likely just a matter of prioritizing the work.

[26] During the SPAR research program, this technology was known as FSS. It has changed significantly since the research program and is now known as Stealth.

Installation The Stealth documentation was well organized and easy to follow. Some steps in the procedure could have been automated, but overall installation was very straightforward.

Configuration Configuration of the Stealth system was very straightforward and was completed without difficulty. The Stealth configuration procedures included a few manual steps that could have been further automated. Ingest was very fast and was performed without any issues. There were additional configuration steps after ingest to distribute appropriate data to the client and third party in preparation for running the system. Specification of policy enforcement was very simple, perhaps owing to the limited capability Stealth has in that area (see Section F.3.4 below).

Operation One early concern was the lack of any logging output from any of their components. The MIT LL team asked the Stealth team to add logging to the client and server adapter components after the risk reduction for connectivity validation and troubleshooting purposes. Their indexing service didn't create any logging output at all. While this was not a problem for the pilot demonstration due largely to the robustness of their software, it would have made troubleshooting problems challenging if any had come up.

Also, once connected, their components were designed to shut down if any peers disconnected for any reason. This effectively meant that all of their components had the same lifecycle. While this was convenient for the pilot demonstration, in a production system the backend services should only shut down if stopped using the platform's service management tools. In an operational context with multiple clients, you certainly would not want the backend services to shut down each time a client disconnected!

F.3.3 Pilot Results

Stealth was easy to setup and configure, and was stable and accurate throughout the demonstration as seen in Table 35. How quickly Stealth handled queries was largely determined by the number of records being returned. Its performance was consistent and fast throughout the pilot as seen in Figure 25. Inserts were processed in batches on a set schedule which made this a relatively quick process as seen in Table 36. The new records would not be reflected in query results until the batch processing ran. Deletes were processed in real time and took effect immediately, deletes took approximately 0.0066 seconds per record (data in Table 37). Stealth's ability to reject queries based on policy was based on black-listing, and therefore Stealth was not capable of enforcing all of the requested policy rules. Stealth's policy enforcement is discussed in the next subsection.

F.3.4 Policy Testing

Stealth builds its query check rules as a set of blacklists and therefore cannot check for the inclusion of a field in a query. Further, we expected that Stealth would have been able to perform range checks, but it struggled with range checks as seen in table 39. On the other

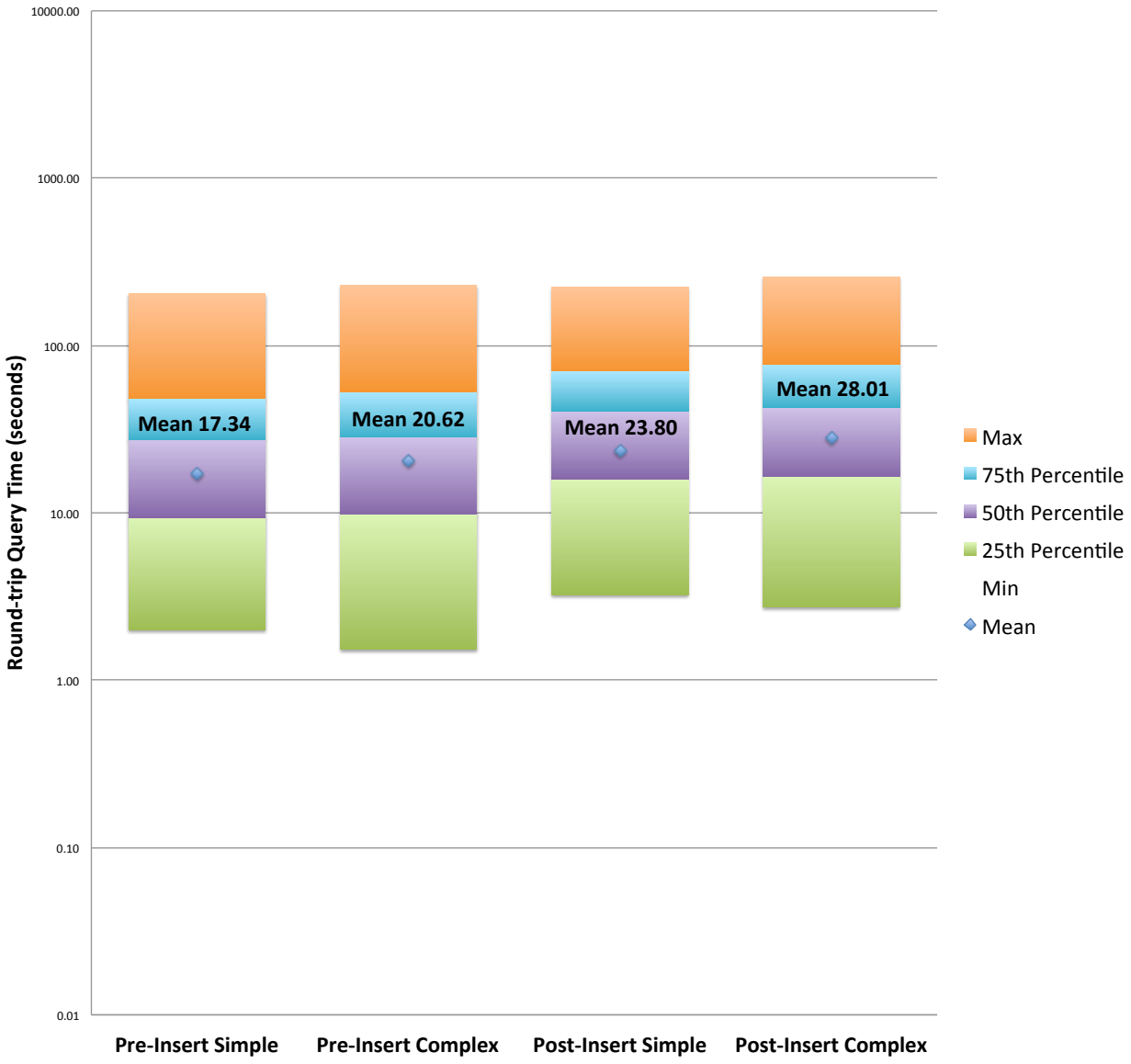


Figure 25: Stealth Pilot Performance. Query response times are split by simple/complex queries and pre/post-insert performance.

	Technology	Overall Mean	Standard Deviation
Pre-Insert	Stealth	18.90	19.00
	MySQL	2.01	6.54
Post-Insert	Stealth	25.80	20.60
	MySQL	2.10	6.61

Table 34: Stealth Performance Summary, time in seconds

Queries Answered	Accurately	Inaccurately	Error
Stealth	199	0	3
MySQL	202	0	0

Table 35: Stealth Query Expressivity, number of queries. Accurate queries returned without error and with the correct results. Inaccurate queries returned without error but did not match the MySQL baseline. Error queries include queries reported as not supported as queries that returned an explicit error.

Records Inserted	10,000	100,000	500,000	1,000,000
Time (s)	3,060	727	5,930	13,900

Table 36: Stealth Insert Performance, time in seconds

Query	Records Deleted	Time(s)
SELECT * FROM base WHERE LastName = 'Rabe';	119	1
SELECT * FROM base WHERE FirstName = 'Sunny';	363	2
SELECT * FROM base WHERE FirstName = 'Hunter';	562	4
SELECT * FROM base WHERE FirstName = 'Will';	1022	6
SELECT * FROM base WHERE FirstName = 'Doug';	2181	13
SELECT * FROM base WHERE FirstName = 'Luke';	2237	16
SELECT * FROM base WHERE LastName = 'Fuller';	2727	18
SELECT * FROM base WHERE FirstName = 'Max';	3200	21
SELECT * FROM base WHERE FirstName = 'Karl';	3948	26
SELECT * FROM base WHERE FirstName = 'Ruben';	4538	30

Table 37: Stealth Delete Times in Seconds. All queries were executed after delete to ensure all records were deleted.

Error rates for valid queries	Stealth
Valid Conjunctive Queries (63)	0%
Valid DNF Queries (101)	1%

Table 38: Error rates for valid queries. Lower numbers are better

Error rates for invalid queries	Stealth
Boolean Field Omitted (167)	55%
Boolean Field Wrong Value (242)	0%
Range Wrong Value (195)	100%
Range Too Big (141)	99%

Table 39: Error rates for invalid queries. Lower numbers are better.

hand, Stealth performed well at accepting valid queries as seen in table 38. It correctly accepted all valid conjunctive queries and most valid DNF queries. Overall, the ability to enforce the desired rules was not good as seen in table 40.

F.3.5 Operator Feedback

In this section we provide feedback about Stealth from operators during the pilot demonstration. Responses to usability questions are shown in Figure 22. The Stealth system had consistent performance and operators found the system very usable. Operators expected performance to scale with the number of records returned and the Stealth implementation matched this mental model. Some operators requested advanced features (such as inner joins) that do not currently exist in any SPAR implementation. This feedback is reflected in the quotes below:

- "This provides a capability I don't have. If it's the fastest thing out there, then I would use it and account for the slowness in staffing/schedule. However, if something faster is available, I'd migrate to that."
- "Easy to use, acceptable amount of time for searches."
- "The technology was easy to use and would definitely be helpful to the community."

Ability to support Tested Business Rules	Stealth
Rule 0 (Allowing Valid Queries)	High
Rule 1 (Inclusion of Boolean Value Set to 0)	Low
Rule 2 (Inclusion of Date Set to Proper Range)	None
Rule 3 (Restriction of Date to Small Range)	None

Table 40: Current ability to enforce Business Rules (see 3.2.6). Based on the mapping between Tables 38 and 39 and the list of Business Rules.

- "Wasn't able to handle a 'not' statement."
- "Need the ability to perform inner joins that would allow faster query time."

F.3.6 Identified Limitations

The technology was easy to set up, configure and maintain. The system was robust and testing was able to proceed smoothly and without incident. The following limitations in the technology were observed by the MIT LL team:

1. The Boolean query functionality was a new feature. Stealth has provided a high-level description of this functionality [38]. The description is reasonable but insufficient to rigorously evaluate the security of this functionality.
2. The policy checks were quite limited. Policy range blacklisting didn't work and more expressivity is necessary for the query structure checks. Stealth's policy check capabilities were very different than those requested for the pilot. The Stealth team had limited time to support the requested rules and made some progress; they believe it is possible to support these rules with additional work [38].